AWS Attendance System

by

Jakub Kierznowski

Submitted in partial fulfillment for the requirements of
TECH 4513 - Senior Project in Information Technology
Spring 2024

Contents

1. Topic
   1.1 Problem
   1.2 Motivation
   1.3 Description
   1.4 Scope

2. Teams and Tasks (Required for Teams)

3. Project Breakdown
   3.1 Architecture
   3.2 Data Analysis

4. Challenges and Takeaways

5. Future

6. Additional Notes

7. References

1. Topic

This is a database management system in AWS (Amazon Web Service) created to connect with and store worker attendance for records of work hours in a noSQL based data storage that is viewable and updatable by the worker themselves and management on the cloud.

1.1 Problem

The improvement or innovation targeted here is moving the database management system from your computers or servers to aws for ease of use, access, and convenience along with security. Many databases or storage systems are kept in computer or company owned servers and systems but this comes with its own issues and problems such as limited space and storage, security can vary depending on real world factors and how good your cyber security set up or team is. Along with this usability of where you can access it, how can you check on information and track traffic and update your system manually to keep up with the ever changing environment plus technology. Moving everything to AWS or any cloud service provider is an improvement but also an upgrade in ease of use, accessibility, security, maintenance, management, structure, reliability, and space.

Going into the problems and issues found in more detail will give a better understanding of why the move is beneficial and an improvement. First issues or problems to discuss are security, space, storage, and pricing since there are some of the main factors that a company will think about when wanting to create systems and set up servers for them. With security you have to worry about your own firewall, how good is your cyber security team, what kind of permission and systems do they have in place or if you're buying a security service, how good is it, and that's another extra payment added onto your setup to pay for. In AWS security is one of their key factors and features mainly focused around their IAM system or permissions, along with built in authentication and security services and systems you don't have to worry about and are easy to manage, all included and affordable and changeable at any time. And you don't have to worry about the physical security of your server since Amazon has it covered and they place many backups around so you can access your systems and be safe.

Spacing is another key factor one has to think about when setting up the system you have to figure out how much spacing/storage you will need to hold all your data and information, get some actual physical space to set up the servers and create an environment with all the safety features needed to run the system yourself such as heating control and cooling, anti fire safety measures. Along with this comes the process of figuring out how you will set up everything to connect to one another and not have wiring and placement issues. After that comes what hardware you will need to make everything work and make sure it all can communicate and work together, plus after all that you need to consider future upgrades and expansion of storage and servers if things keep growing or you run out of space. At the end of the day setting up a server room or servers is a time consuming hard task that takes many factors into consideration and requires a lot of problems solving and planning along with money to pay for all these aspects needed for it to run. With AWS most of these issues get solved for you and make it all easier in the end. You can set pricing limits and how much spending is done by the system to store your data. You don't have to set up the physical servers or worry at all about that aspect along with having access to the service and at any time anywhere. All the traffic interactions and inputs are recorded and tracked to keep you safe and to keep watch for any irregularities. There are multiple servers world wide with many back ups you can make in multi regions and areas allowing your data to be safe and avoid risk of damage or loss. Lastly you can edit and change size of storage and limited data inputs into it along with setting that you pay as you go instead of just a budget over time that you can pass, but with this option you can set a ceiling of how high the money can go.

All these improvements are made by using AWS over setting up your own system from scratch. There will of course be some learning issues at first with having to figure out a new system and how it all works, but once it's all figured out by your workers and management everything will be working and going smoothly with backups in place and your future set and guaranteed.

1.2 Motivation

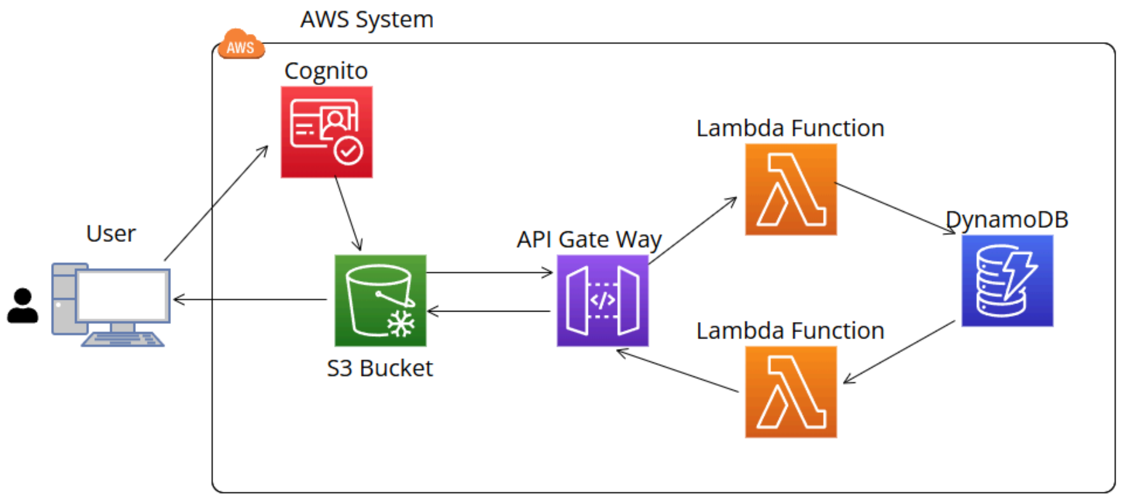
The initial inspiration for this project came from the ever changing and evolving work environment that has greatly changed since the whole COVID epidemic. Most work forces and companies used to just have their own things and have it all in the office for them to work and manage but as things have evolved and changed over the years new perspectives and cost efficiency and money saving have led to more interest and innovation in these areas. With more people working from home and everyone needing access to information systems and other systems the use of cloud storage and systems is the future and it's a continuously growing and expanding market, and field. With this we can see a clear path for how future work spaces and requirements can look like.

Seeing this trend or growth that is happening makes me wish to prepare myself for the future and what it holds in my chosen field of study and work. Along with this I actually have a joy and interest in AWS itself. I enjoy working with the system itself and fighting out how everything works and is connected. This improves myself and gets me ready for the future along with pushing me to improve and pursue a passion or interest of mine. This takes out two birds in one stone as they say so how could i not pursue and want to do a project involving AWS. As for why this type of project itself, well one of the biggest aspects of AWS itself is storage or holding things virtually for you along with a containing system. Storing workers data just seemed like an obvious aspect or idea to pursue since it's a common thing that gets done and used by all companies to keep track of workers and work hours for pay reasons and performance statistics for maybe future bonuses for the employee if they did a great job. Saving data to a table and retrieving and editing it also just seems like one the basic fundamental aspects of how most work forces and environments are, and a needed skill to make sure information and data operates properly at the job. And being able to link all these different systems and learning the proper security needed for them to work in the end with each other using IAM was a mandatory skill needed for future uses of AWS or other cloud systems since they mostly operate the same way just using different naming and terminology. Along with this knowing how Lambda works and API gateway which will be explained and further talked about later in the report are very common and always used functions and systems within AWS for the everyday user. This all in the end gives me a good building block to further my research and understanding of AWS to eventually go for a certificate and qualification in AWS in the future since it will look good for my job searching and resume.

1.3 Description

Before going into great length over the details and aspects of my system, here is an image of the system to get an idea of all the interconnecting parts.

Figure 1. The Attendance system structure and layout

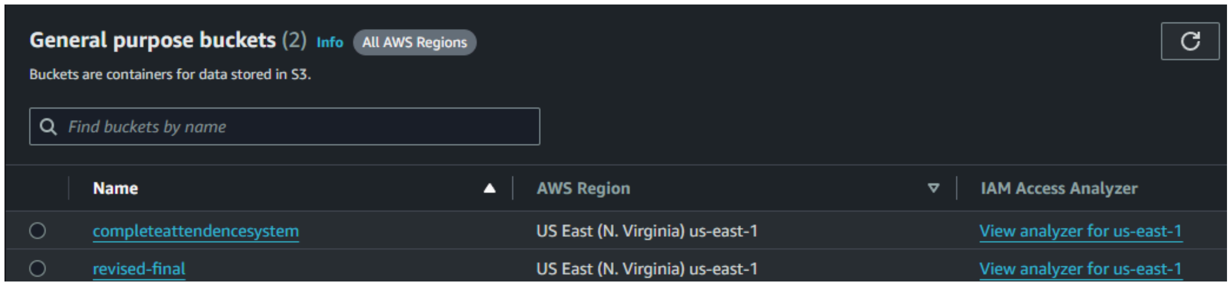


The whole system starts in AWS (Amazon Web Service) it's all stored and coded there, so the actual user's computer contains nothing and there's no space or storage being taken up unless you download it manually. There are exactly 6 components to this system and project the IAM requirements, the login and credentials verification, the actual code and site storage, the API gateway connection through lambda, and the Database table/storage. We can now give a brief in detail breakdown and explanation of how it all works and runs then further in the breakdown section we can go into more detail of the technical aspects of the whole project in general and its individual components. It starts with the user themselves and their machines to interact with the system itself.

First the user needs to possess the proper permission to access and run the functions needed and interact with all the different system that run on AWS for that they need the proper IAM permission which are like a digital key that allows the user to run the functions and pass the proper data into the table for storage, without this nothing will work and you will just get errors. Now this would usually require you to log into AWS with your user specific credentials and have your account have access granted by administration to use the different aspects of the attendance system, but by linking the IAM user account to the next portion we can do this by skipping the AWS login needed. The so-called next portion we link the IAM and its permission to is a Cognito user database. Here you can create a user database with user names and password that only people with the right permission can access or accept new users into the system data bank. After that set up and link the proper IAM user to the users in the data bank or to save time and make it easier link the list to a single IAM user that you can label whatever “attendance” as an example that has the permission to just use the attendance system so all those people on the list can then do what they need to login and interact with the system. What Cognito is doing in this case is giving you something like a cookie with the credentials and permission of that IAM user temporarily for whatever time you set without having to login to AWS; this keeps things contained and protected, preventing anyone from doing anything else. The actual login info for the IAM accounts is not leaked or used, you just have access to its special key that only allows for use of the attendance system and nothing else.

After going through all that you can finally login and access the site itself to interact with the website or page to plugin information. This whole site in all along with this style sheet is stored in the S3 bucket which holds them all and hosts the site from their but first specific settings had to be set to allow public viewing and access form outside of an approved AWS IAM account, basically allowing login attempts without having to go threw aws console as long as you have the link to try as shown below in figures 2 and 3.

Figure 2. S3 bucket

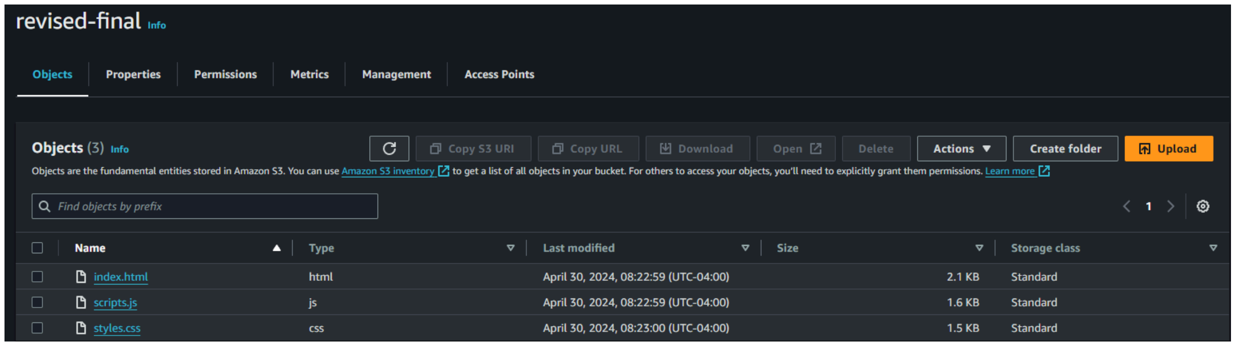


General purpose buckets (2) Info All AWS Regions

Buckets are containers for data stored in S3.

Name	AWS Region	IAM Access Analyzer
completeattendencesystem	US East (N. Virginia) us-east-1	View analyzer for us-east-1
revised-final	US East (N. Virginia) us-east-1	View analyzer for us-east-1

Figure 3. inside project bucket


revised-final Info

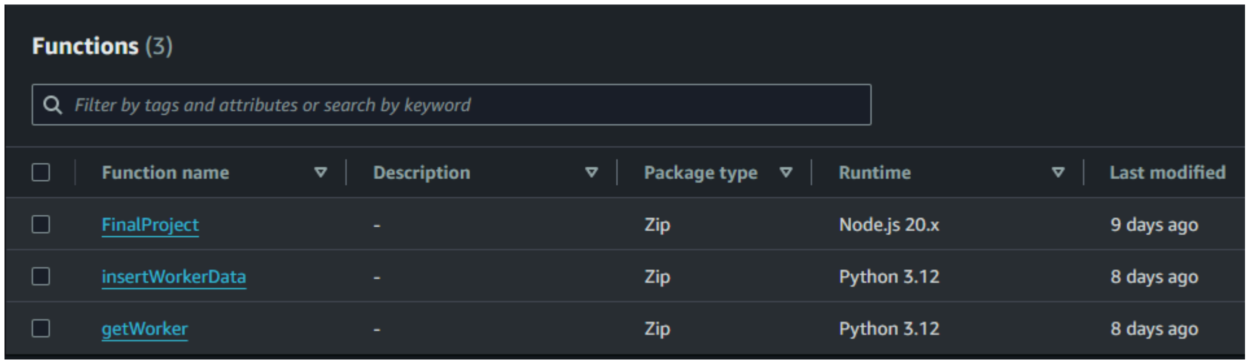
Objects (3) Info

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

Name	Type	Last modified	Size	Storage class
index.html	html	April 30, 2024, 08:22:59 (UTC-04:00)	2.1 KB	Standard
scripts.js	js	April 30, 2024, 08:22:59 (UTC-04:00)	1.6 KB	Standard
styles.css	css	April 30, 2024, 08:23:00 (UTC-04:00)	1.5 KB	Standard

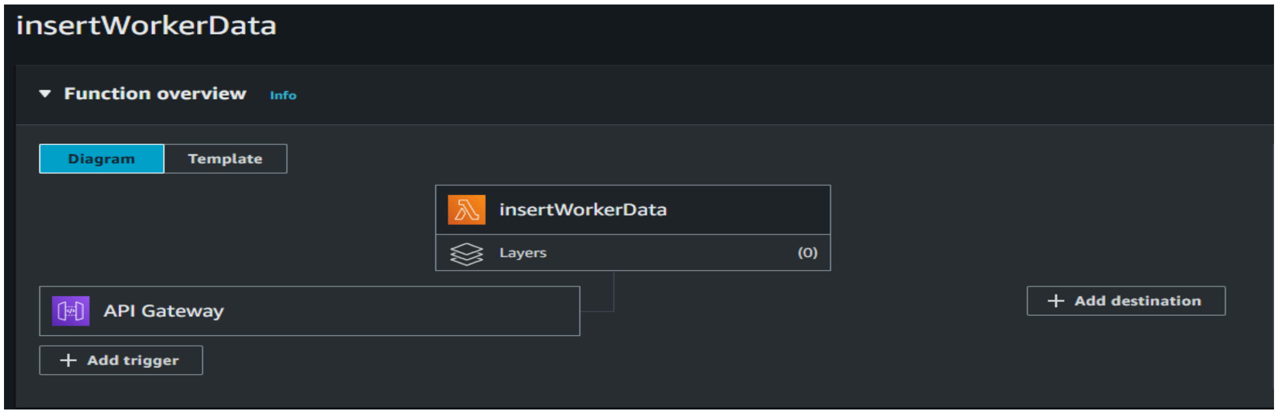
Next comes the API gateway and lambda which is used to actually communicate between the database and user interface, the API gateway makes sure that only that specific HTTP or site can connect to this database and secures/protects the data and information traveling between them like a direct link tunnel. No outside source cna communicate with this database or call its functions since it's not linked directly. While API gateway does this and calls the function to the database the Lambda does the actual talking to and calling plus execution of functions. Lambda is a serverless computing platform that basically allows me to run code and execute functions in response to specific actions or signals being given that were pre-programmed. Here its executing some functions in python to add to the database and another to get from the database as shown below in figure 4 and 5.

Figure 4. the Lambda functions


Functions (3)

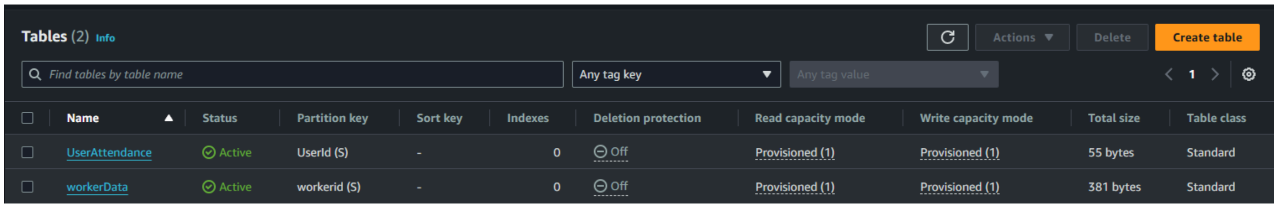
Function name	Description	Package type	Runtime	Last modified
FinalProject	-	Zip	Node.js 20.x	9 days ago
insertWorkerData	-	Zip	Python 3.12	8 days ago
getWorker	-	Zip	Python 3.12	8 days ago

Figure 5. the connection between lambda and API gateway



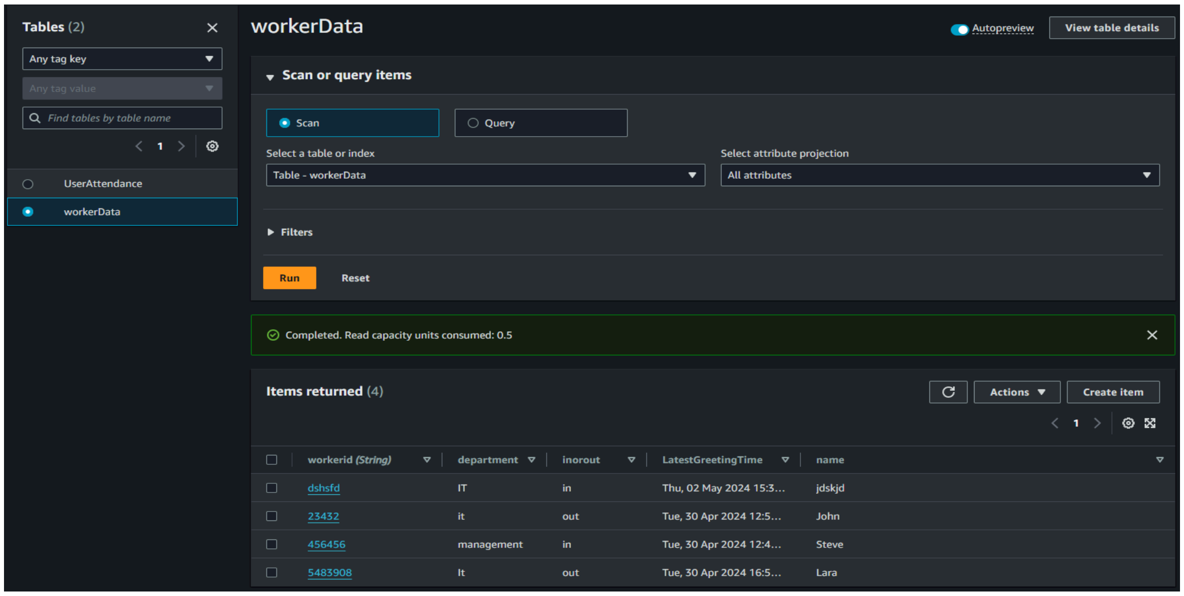
Finally we have the DynamoDB database which is used to store all our workers data in a table readable formatting that anyone with the right permission can access. This is a NoSQL database that stores our data and provides encryption and safety to any data stored or taken out of it. Table example of console view in figures 6 and 7.

Figure 6. DynamoDB Tables



Name	Status	Partition key	Sort key	Indexes	Deletion protection	Read capacity mode	Write capacity mode	Total size	Table class
UserAttendance	Active	UserId (S)	-	0	Off	Provisioned (1)	Provisioned (1)	55 bytes	Standard
workerData	Active	workerid (S)	-	0	Off	Provisioned (1)	Provisioned (1)	381 bytes	Standard

Figure 7. Example of managements table editing and viewing



Select a table or index: Table - workerData

Select attribute projection: All attributes

Completed. Read capacity units consumed: 0.5

workerid (String)	department	inorout	LatestGreetingTime	name
dshsfd	IT	in	Thu, 02 May 2024 15:3...	jdskjd
23432	it	out	Tue, 30 Apr 2024 12:5...	John
456456	management	in	Tue, 30 Apr 2024 12:4...	Steve
5483908	It	out	Tue, 30 Apr 2024 16:5...	Lara

1.4 Scope

The main focus on or scope for the project was to create a database that the user/worker can interact with and input their information and have it record their attendance for work that day and keep track of attendance. The actual login function was added later for security reasons and to have some form of access into the system outside of AWS IAM. Another key factor of scope was pricing. This entire project was meant to be done for free without any additional cost needed, I'm currently not in the best financial situation and neither is my family so saving money and keeping costs down is the main goal. Plus this keeps in mind the whole workforce mind set of having a set budget and learning not to go over and get things done in it. This system will not do much of anything else other than allow login and verification of credentials, allow access into a website, allow input and retrieval of data from a database and only that to keep security in the best form blocking access to all other unnecessary things. Features and how things look and operate changed a lot over the process but that will be covered later in challenges and takeaways. This is a very narrow, limited, and contained scope or project relying on the most efficient and functional code and use of functions plus systems.

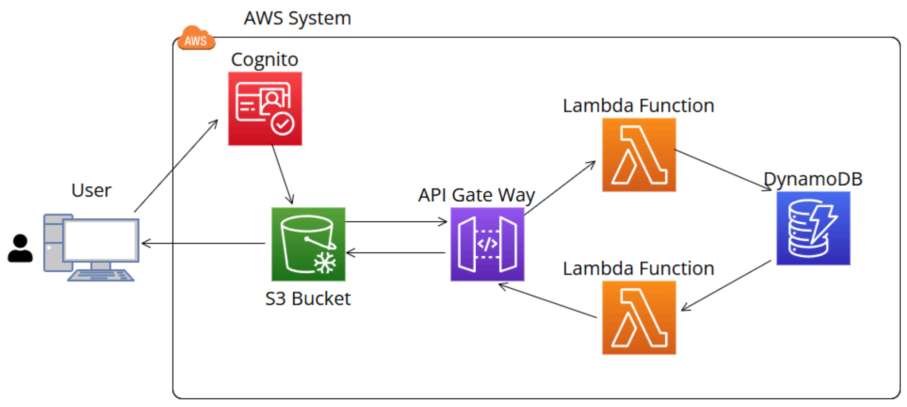
2. Teams and Tasks (Required for Teams)

Since there's only me for this project I will say Jakub Kierznowski did all the research, study, implementation, code writing, testing, debugging, and writing for the final paper.

3. Project Breakdown

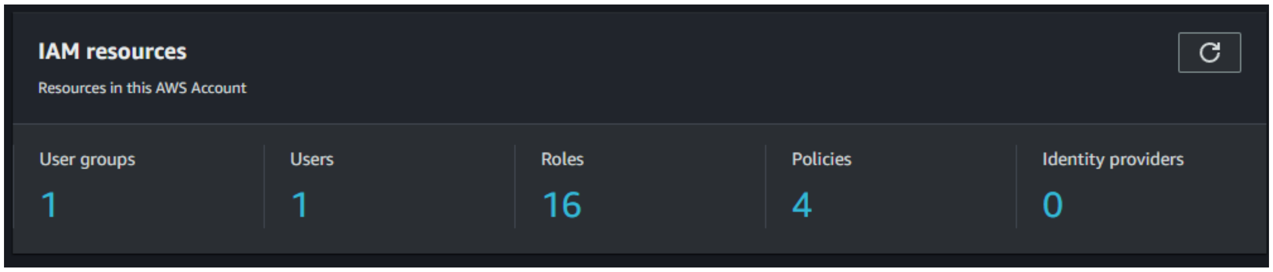
3.1 Architecture

Figure 7. The Attendance system structure and layout



As shown before but now again in figure 7 we have the structure and layout of our system and its parts now we can really break down each individual component and how they connect properly. The system itself is found on AWS the Amazon Web Service which is a on demand cloud based service with cloud computing platform and API's (Application programming interface) which runs thousands of ISV's or Independent software vendors. This means that the system we are running does not exist on any of our machines physically but is stored somewhere else on a server run by Amazon, allowing quick and easy access from anywhere to our programs and systems. The first Major system we need to discuss is the IAM credentials and users. This is the backbone of AWS and how it keeps things secure and working for companies and users. The first step was setting up a root account which has access to everything in AWS and is the original account that none can bypass or supercede, and form their we make our first IAM user to have access and work on our systems they have a unique username and password but are all linked to an account number given to the root account on making it and has to be listed in the login to gain access to the console. after making the user they are given an ARN (Amazon Resource Name) they are fundamentally important and to be remembered its specifies AWS resources to call or use. This is how you know what access the service will have, what options are available to them and how to connect to individual components in AWS with one another It's a key. Along with this key comes another called the Access Key which is a unique code or password needed if you want to identify your IAM user and access any services or connect to AWS from an outside source to inform it if you can even use whatever resource you wish to. Now to actually access anything in AWS your IM User needs Policies attached to them by the root or an admin to user services, Policies give access to functions or calls that users can make or to even allow them to open and see a database. Now instead of attaching 30 different policies to a user we take these Policies and group them together into a Role which contains all these policies then we link the role to a user to give them it all. And if you really need to you can make a User group put the users into the group and link that group to a Role. This is the start of our system we now have access to everything and
can work on things.

Figure 8. IAM Users

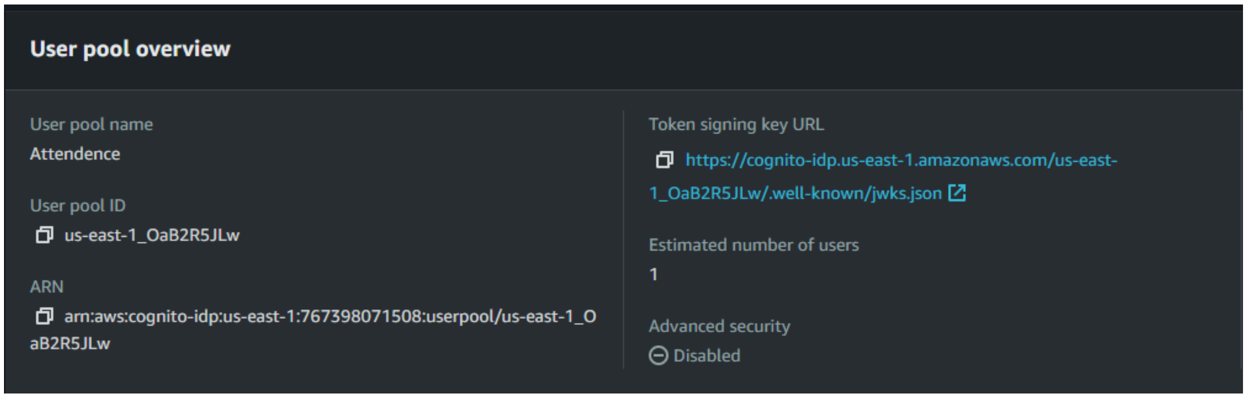

IAM resources

Resources in this AWS Account

User groups	Users	Roles	Policies	Identity providers
1	1	16	4	0

Next we must talk about the Cognito system and how it lets us access our site and gives us the IAM to make everything work. Amazon Cognito is a CIAM service or Customer IAM that implements identity and access management for users/customers of a site which scales. It works by creating a database of users separate from everything else which you can set limitation on password, user name, ect and set it up to either allow sign up or not and only admins can access and physical add a user which then they need to verify themselves somehow my email or phone, you can include two factor authentication even if wanted. Now that we created a user and added them to our system and they have their credentials we now have to input into our script on our site a small section to allow access to the table and info using its User pool ARN and user pool ID. But now that we're on our first actual tool outside of IAM we do need to first talk about another security measure and factor when designing and coding everything as shown in the ARN and pool ID in figure 9 below the region lock.

Figure 9 Cognito User pool credentials



User pool overview

User pool name
Attendence

User pool ID
us-east-1_OaB2R5JLw

ARN
arn:aws:cognito-idp:us-east-1:767398071508:userpool/us-east-1_OaB2R5JLw

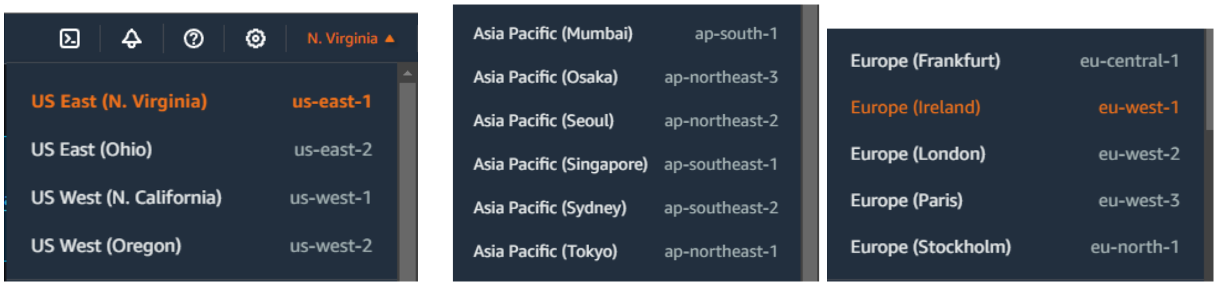
Token signing key URL
https://cognito-idp.us-east-1.amazonaws.com/us-east-1_OaB2R5JLw/.well-known/jwks.json

Estimated number of users
1

Advanced security
Disabled

As seen above the ID and ARN have a specific line listing us-east-1 and that because AWS is divided in multiple regions and zones you need to consider that they are constrained and isolated but can be connected as shown in figure 10 below.

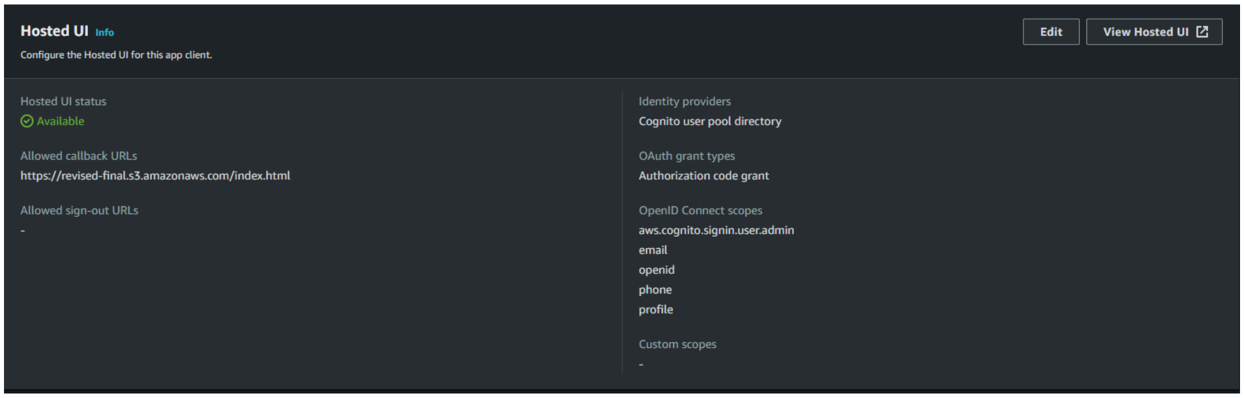
Figure 10 AWS Regions (only some not all)



US East (N. Virginia)	us-east-1
US East (Ohio)	us-east-2
US West (N. California)	us-west-1
US West (Oregon)	us-west-2
Asia Pacific (Mumbai)	ap-south-1
Asia Pacific (Osaka)	ap-northeast-3
Asia Pacific (Seoul)	ap-northeast-2
Asia Pacific (Singapore)	ap-southeast-1
Asia Pacific (Sydney)	ap-southeast-2
Asia Pacific (Tokyo)	ap-northeast-1
Europe (Frankfurt)	eu-central-1
Europe (Ireland)	eu-west-1
Europe (London)	eu-west-2
Europe (Paris)	eu-west-3
Europe (Stockholm)	eu-north-1

When setting up different apps and tools you need to specify a lot of what region to set the data or table in so it will be stored in that region's data and you can access it from another region. This an extra security measure to protect the wide system in case something happens in Europe it will not affect the US servers, also as another safety measure you can set up backups in these other regions and have traffic directed to that if something crashes in the main. So when setting up tools like Cognito and all others going forward you need to consider and specify the region of use for things to work properly, IAM users do not follow the region lock system; their available regions of interactions depend on the Root account and where it's made and has access to. Back to Cognito now that its set up we link our user pool to a Role with policies and an IAM user since when you log in using Cognito the logged in user temporarily is given and generated IAM credentials matching those of the Role linked and IAM user that run out after a set time allowing you to use the system without having to log into the AWS console. After this we just have to list what links the user gets taken to or has access to while having those credentials. This would be our home page or index where we can add or change the attendence as shown in figure 11 below and you're given a unique login url to use.

Figure 11. Host sites and redirects



Configure the Hosted UI for this app client.

Hosted UI status
Available

Allowed callback URLs
https://revised-final.s3.amazonaws.com/index.html

Allowed sign-out URLs
-

Identity providers
Cognito user pool directory

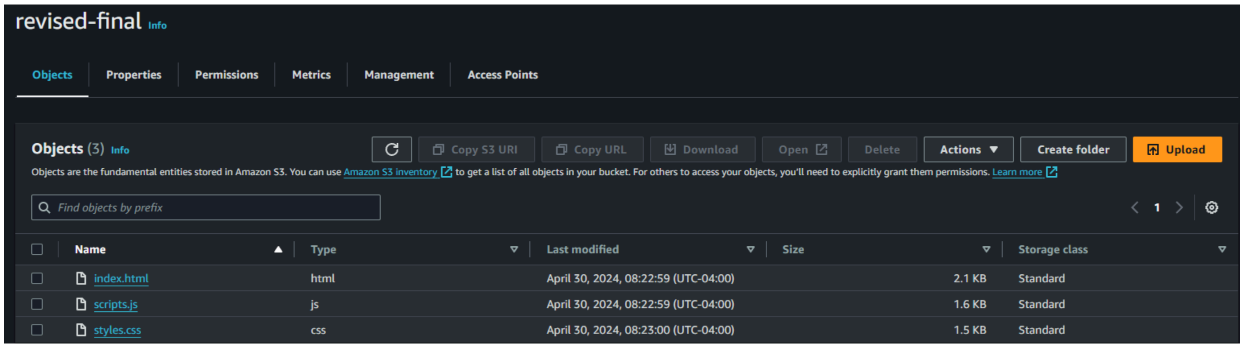
OAuth grant types
Authorization code grant

OpenID Connect scopes
aws.cognito.signin.user.admin
email
openid
phone
profile

Custom scopes
-

Now that we are done with Cognito we move on to the S3 bucket and explain its purpose and functionality. AWS S3 bucket is as the name says a bucket used to store things or more specifically an object oriented bucket to store objects in this case our websites and its code along with its js files which we will use to communicate to our database and add or update it as in figure 12.

Figure 12, S3 Bucket



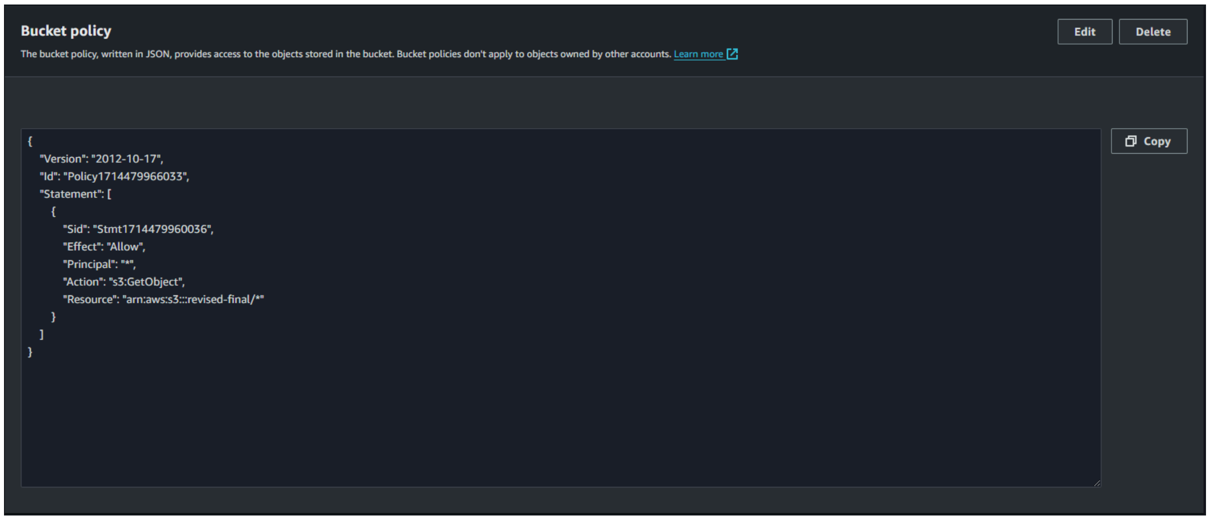
Objects (3) Info

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

Name	Type	Last modified	Size	Storage class
index.html	html	April 30, 2024, 08:22:59 (UTC-04:00)	2.1 KB	Standard
scripts.js	js	April 30, 2024, 08:22:59 (UTC-04:00)	1.6 KB	Standard
styles.css	css	April 30, 2024, 08:23:00 (UTC-04:00)	1.5 KB	Standard

But just implementing our code into the bucket for storage does not make it work right away as a host for our website. I had to first link it to the proper roles and IAM credentials to communicate and perform actions, then after that we had to set some settings to make it a static website. If you just tried to access your website after putting it all into the bucket you would not be able to access it, you would be blocked. I had to first unblock all public access which is be default blocked, then afterwards I had to edit the ACL or access control list to approve reading and writing on the site, and then after all that i had to write a custom bucket policy to allow it to grab and get objects from the bucket to outside it as shown in figure 13 down below.

Figure 13. Custom Bucket Policy



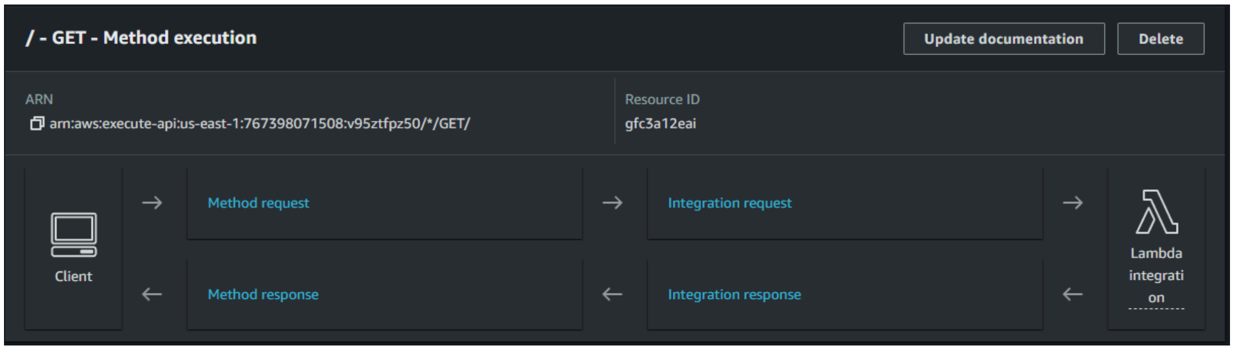
The bucket policy, written in JSON, provides access to the objects stored in the bucket. Bucket policies don't apply to objects owned by other accounts. Learn more

```
{
  "Version": "2012-10-17",
  "Id": "Policy1714479966033",
  "Statement": [
    {
      "Sid": "Stmt1714479960036",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::revised-final/*"
    }
  ]
}
```

now that the policy is in place we need to go into the setting again and set it to allow static website hosting, and finally we need to select all the codes and objects in the bucket and in actions option we need to select make public using ACL and now the site is deployed and accessible for use and can communicate with the rest of AWS even when not logged in to the console.

Then we move to the API gateway which will connect our lambda functions to our database and website and help them communicate and talk to each other through our gateway. API Gateway is a service for creating, publishing, maintaining, monitoring, and securing REST, HTTP, and WebSocket API at any scale for the user's needs. This is a fundamental and very important tool when using AWS since it helps connect most projects and tools in AWS in a secure, and safe manner with great monitoring and handles data, traffic, and heavy loads of information can be scaled to how much you need. But to make our lambda functions work and to communicate to the database we have to set up the methods used by the API gateway, this being our GET method and our POST method. This is using the HTTP GET and POST methods to request information and to add it to the database. AS you can see down below in figures 14 and 15 they show the execution and mapping of how it's done.

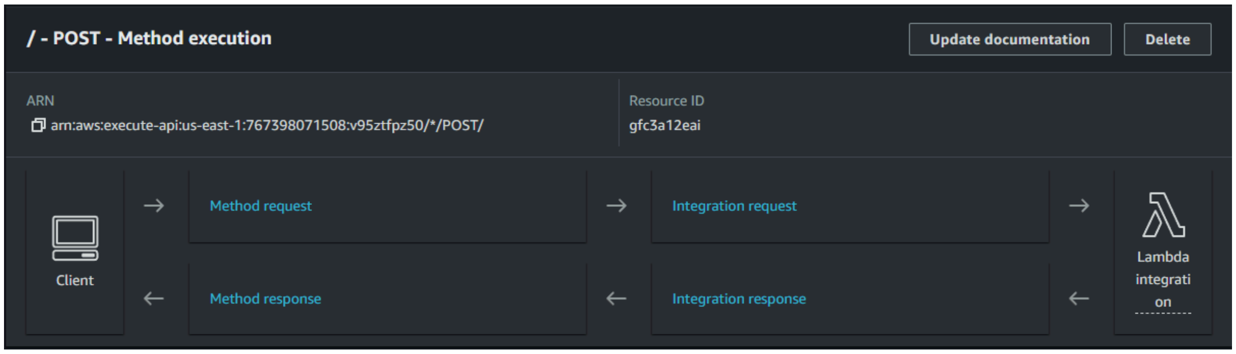
Figure 14. Execution of GET method



ARN
arn:aws:execute-api:us-east-1:767398071508:v95ztfpz50/*/GET/

Resource ID
gfc3a12eai

Figure 15. Execution of POST method

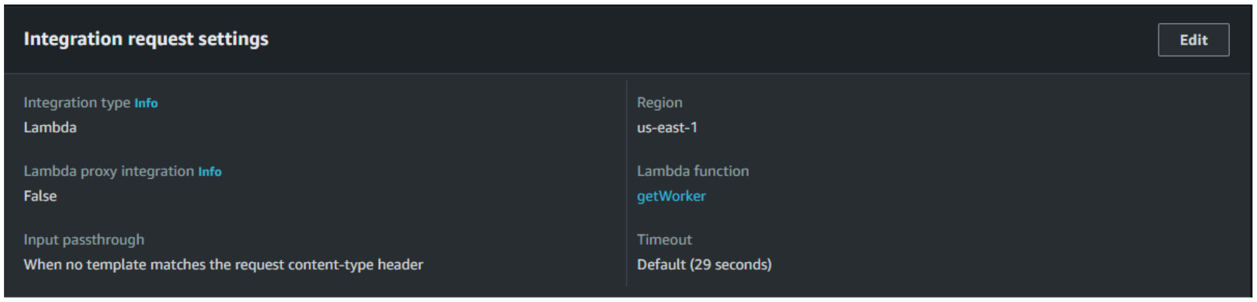


ARN
arn:aws:execute-api:us-east-1:767398071508:v95ztfpz50/*/POST/

Resource ID
gfc3a12eai

As shown above in figures 14 and 15 each method has its own ARN code and shares a resource ID. The ARN code is to execute that specific methodology while the resource ID is the same because they are under the same API labeled worker to function together and being able to be called at the same time using a single code instead of having to individually call each method with separate ARN calls. Now if we click on the integration request on the GET method and POST method you will get the display shown in figure 16 and 17 to show you the setting of the gateway method.

Figure 16. API Gateway GET method integration request



Integration type Info
Lambda

Lambda proxy integration Info
False

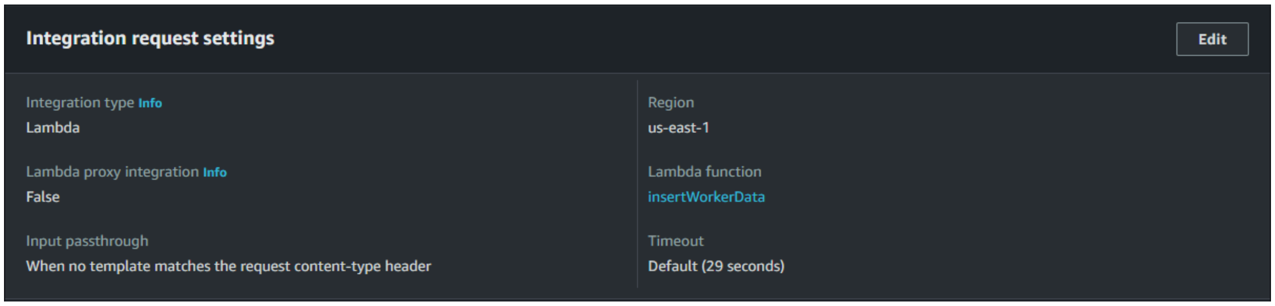
Input passthrough
When no template matches the request content-type header

Region
us-east-1

Lambda function
getWorker

Timeout
Default (29 seconds)

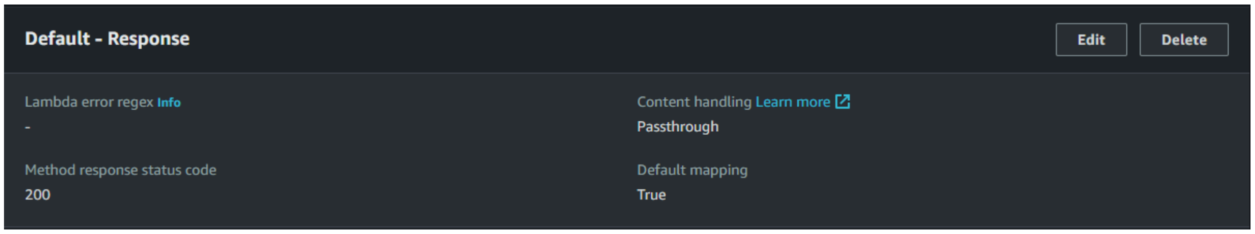
Figure 17. API Gateway POST method integration request


Integration request settings

Integration type	Region
Lambda	us-east-1
Lambda proxy integration	Lambda function
False	insertWorkerData
Input passthrough	Timeout
When no template matches the request content-type header	Default (29 seconds)

You can see that it's integrated and connected to a lambda function. It has a Region setting, a timeout function, and it is calling a specific Lambda function and is different for each one. We will get to those two functions in the next part when we talk about Lambda. As stated previously you have to set a region for each tool/app on AWS since they are divided and contained in those ones specifically and cant really communicate to it from outside the specified region. Each METHOD is connected to a specific Function to perform their appropriate task for either getting ro posting data and if they're an error it will self terminate the request in 29s, and as shown in figure 18 below you can see that if all goes well on the console and response code you should receive back a 200 status code which is common and used practice for testing on AWS for responses.

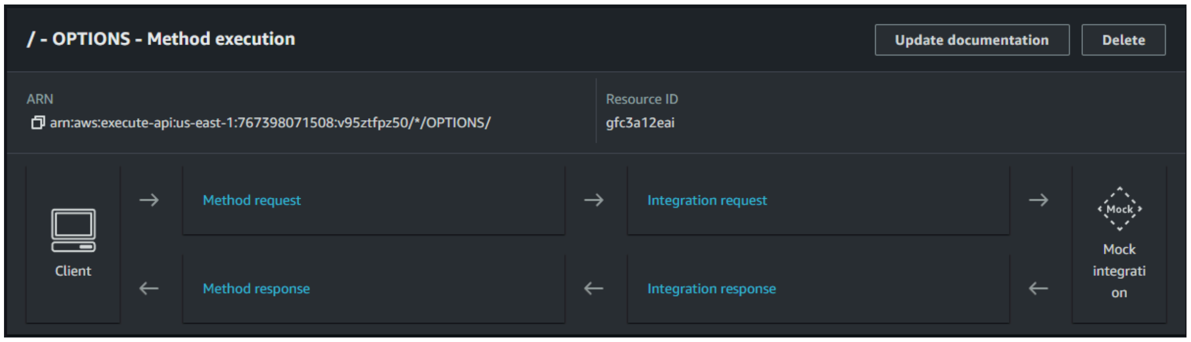
Figure 18. API Gateway integration response


Default - Response

Lambda error regex	Content handling
-	Passthrough
Method response status code	Default mapping
200	True

Now that we have connected our API gateway and connected it to our lambda, if you try to work the website it will still not work because we still need to input two more steps for it all to function and work with one another. We are still missing the Options method of execution which basically boils down to what permission and what systems/tools can this API gateway talk to and what kind of data and functions can it transfer. Shown in figure 19 below the structure and set up of the OPTIONS method which is still located under the same API as can be seen from the Resource ID.

Figure 19. API Gateway OPTIONS Method execution

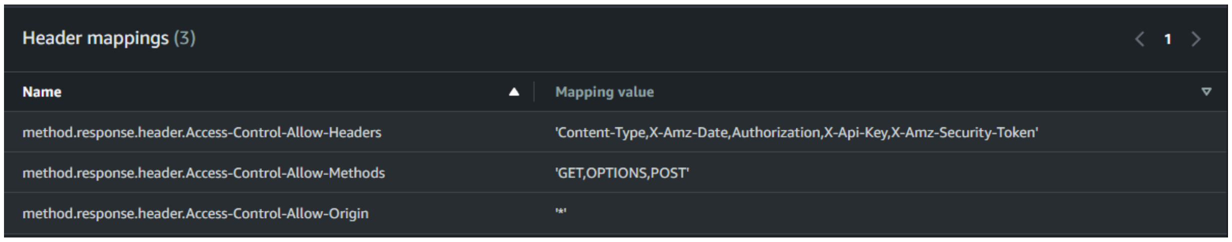

/ - OPTIONS - Method execution

ARN: arn:aws:execute-api:us-east-1:767398071508:v95ztfpz50/*/OPTIONS/

Resource ID: gfc3a12eai

The Mock Integration listed in the figure above is a feature that enables API developers to generate API responses from API Gateway directly, without the need for an integration backend since usually this whole process and system would require some backend setup. But as stated before AWS makes things easier and far more reliant and convenient to use and integrate. The most important factor of the OPTIONS Method is the header mapping which is similar to a Policy that is applied to an IAM user showing what it has access to and what the API can do as shown in figure 20 below.

Figure 20. API Gateway OPTIONS Method Header mapping

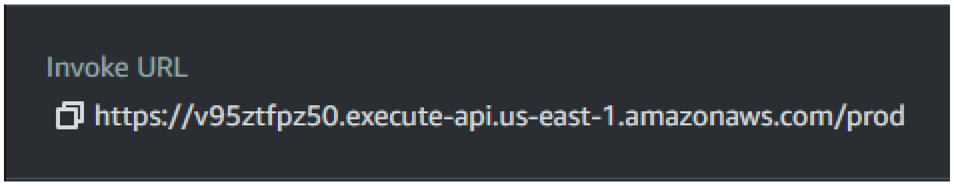

Header mappings (3)

Name	Mapping value
method.response.header.Access-Control-Allow-Headers	'Content-Type,X-Amz-Date,Authorization,X-Api-Key,X-Amz-Security-Token'
method.response.header.Access-Control-Allow-Methods	'GET,OPTIONS,POST'
method.response.header.Access-Control-Allow-Origin	'*'

Shown above in figure 20 the access control allow headers method is looking for the specific authentication and security tokens that were generated from the IAM user and applied to the user who logged into the site showing they have access to do what they want on the site. The Methods that can be used and executed are listed below that meaning you have the IAM permission to GET, POST, and allow the OPTIONS to work in the first place. As for the Origin that specifies what kind of data it will accept and the "*" input just allows all responses but you can fine tune this for better security.

This also plays a big factor in CORS which is important for the communication process and what we had to set up next to make sure our bucket in S3 could actually call GET and POSt methods. CORS is Cross-origin resource sharing and it basically allows the javascript which is used on the to pass through the API gateway and have access to the AWS services since regularly it would be blocked without this. If you look back at Figure 13 you can see we input into the principle "*" allowing all data transfer and contact between the site and our AWS service. Now that everything is set, that API gateway has to be launched/deployed to actually work and be in effect because if it's not none of the requests and functions will work or reach our database and we will just get an error message. Once its deployed it can be named whatever you want but a new URL link to invoke the API gateway that was just deployed will be given and needs to be included in the website code as shown in figures 21 and 22 below.

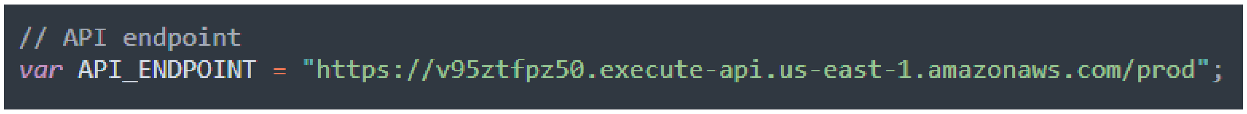
Figure 21 Deployed API Gateway link



Invoke URL

https://v95ztfpz50.execute-api.us-east-1.amazonaws.com/prod

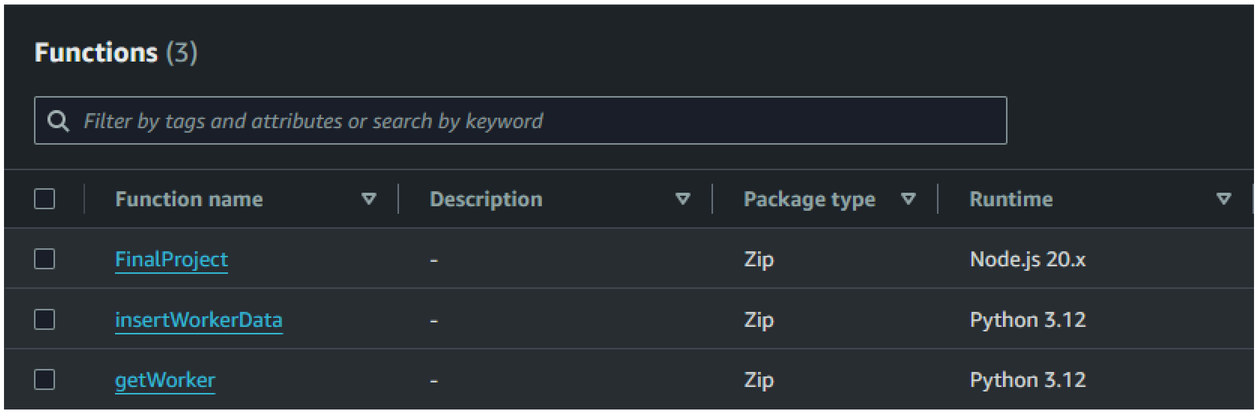
Figure 22. Input code in scripts.js



```
// API endpoint
var API_ENDPOINT = "https://v95ztfpz50.execute-api.us-east-1.amazonaws.com/prod";
```

Since the API Gateway is set up now we can move on to the Lambda functions and Lambda itself. Lambda is an event-driven, serverless computing platform that is designed to enable developers to run code without provisioning or managing servers , and executes code in response to events while automatically managing the computing resources required by that code. To make our system work two functions were needed, one to POST or insert data into the table, and another to GET or retrieve data from the table as shown in figure 23 below and it's all done through python version 3.12.

Figure 23. Lambda Functions



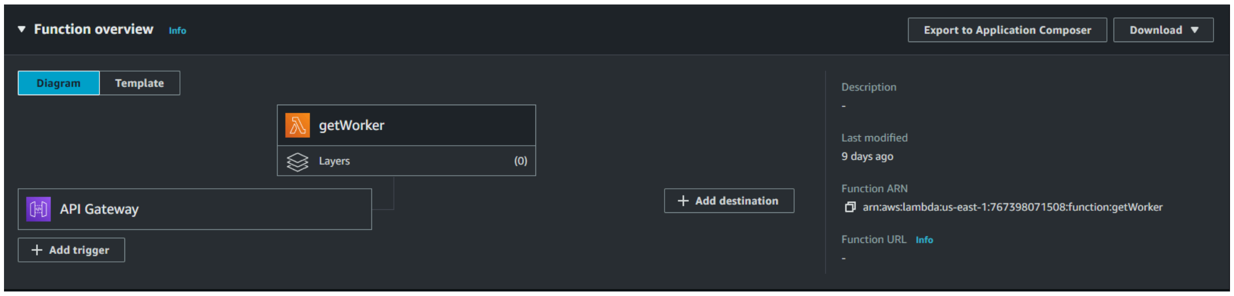
Functions (3)

Filter by tags and attributes or search by keyword

Function name	Description	Package type	Runtime
FinalProject	-	Zip	Node.js 20.x
insertWorkerData	-	Zip	Python 3.12
getWorker	-	Zip	Python 3.12

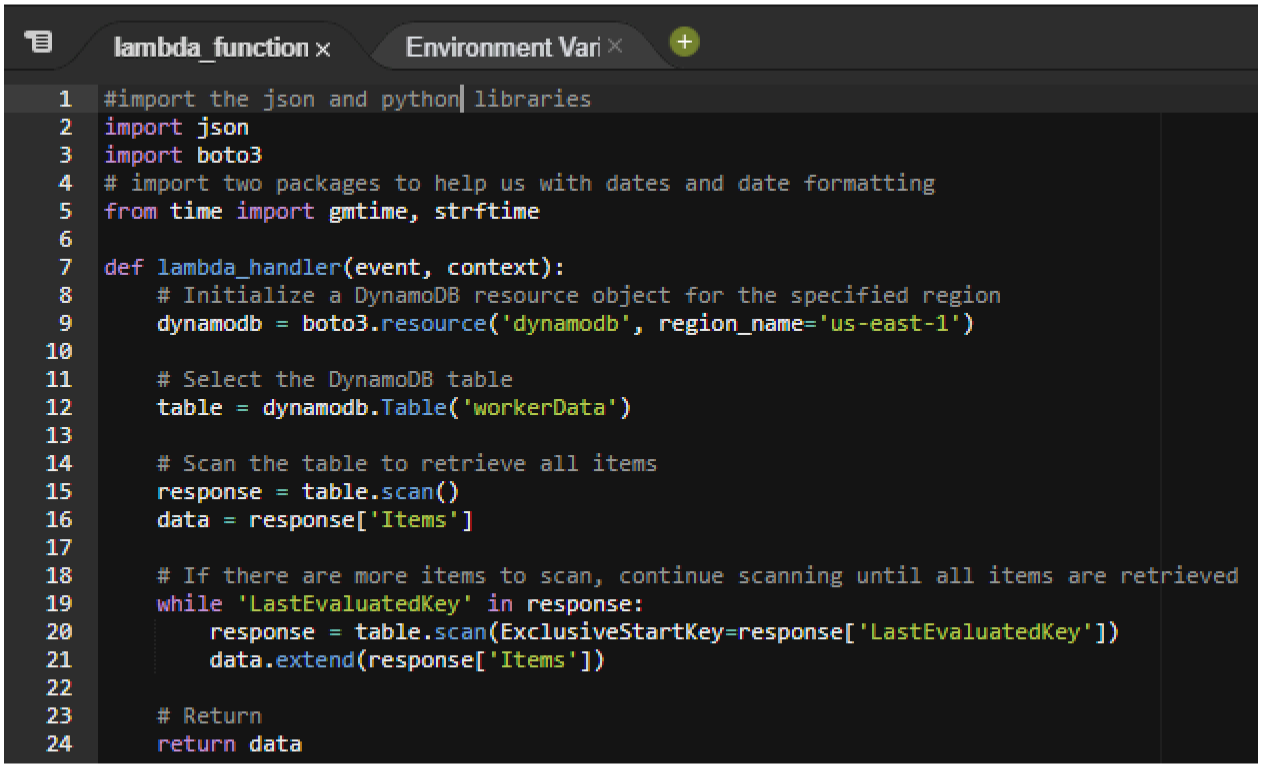
And from figure 23 you can see its stored as a Zip file while running on Python these are setting you must select and specify when first setting up the function and shown in figure 24 you can see the overview chart to show what its connected to, and in figure 25 we see our function code with some small comments explaining what that portion is doing.

Figure 24. Lambda Get Function Overview



Description
-
Last modified
9 days ago
Function ARN
arn:aws:lambda:us-east-1:767398071508:function:getWorker
Function URL Info
-

Figure 25. Lambda Get Function Code

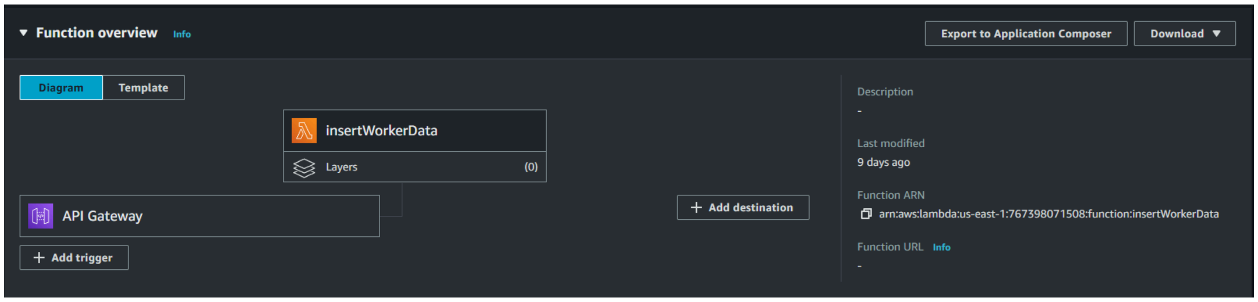


```python
#import the json and python libraries
import json
import boto3
# import two packages to help us with dates and date formatting
from time import gmtime, strftime

def lambda_handler(event, context):
    # Initialize a DynamoDB resource object for the specified region
    dynamodb = boto3.resource('dynamodb', region_name='us-east-1')

    # Select the DynamoDB table
    table = dynamodb.Table('workerData')

    # Scan the table to retrieve all items
    response = table.scan()
    data = response['Items']

    # If there are more items to scan, continue scanning until all items are retrieved
    while 'LastEvaluatedKey' in response:
        response = table.scan(ExclusiveStartKey=response['LastEvaluatedKey'])
        data.extend(response['Items'])

    # Return
    return data
```

Here we have our lambda function code using python and json to retrieve data from the table. And once again we must specify the region we are using and what region the DynamoDb table is made in and set in us-east-1. and after that we just call the table by its name, have it scan the content and grab all the listings. DynamoDB uses a prime key system or a set main value it will look at when storing data that can not be repeated, for our table it's a worker ID so we need to have the scan look at all the keys available and go to the last one and give back all its listings instead of just one or a specific one. After all this I had to connect an IAM Role to the lambda function so it can have the permission to communicate with the API Gateway and the DyanmoDB database.

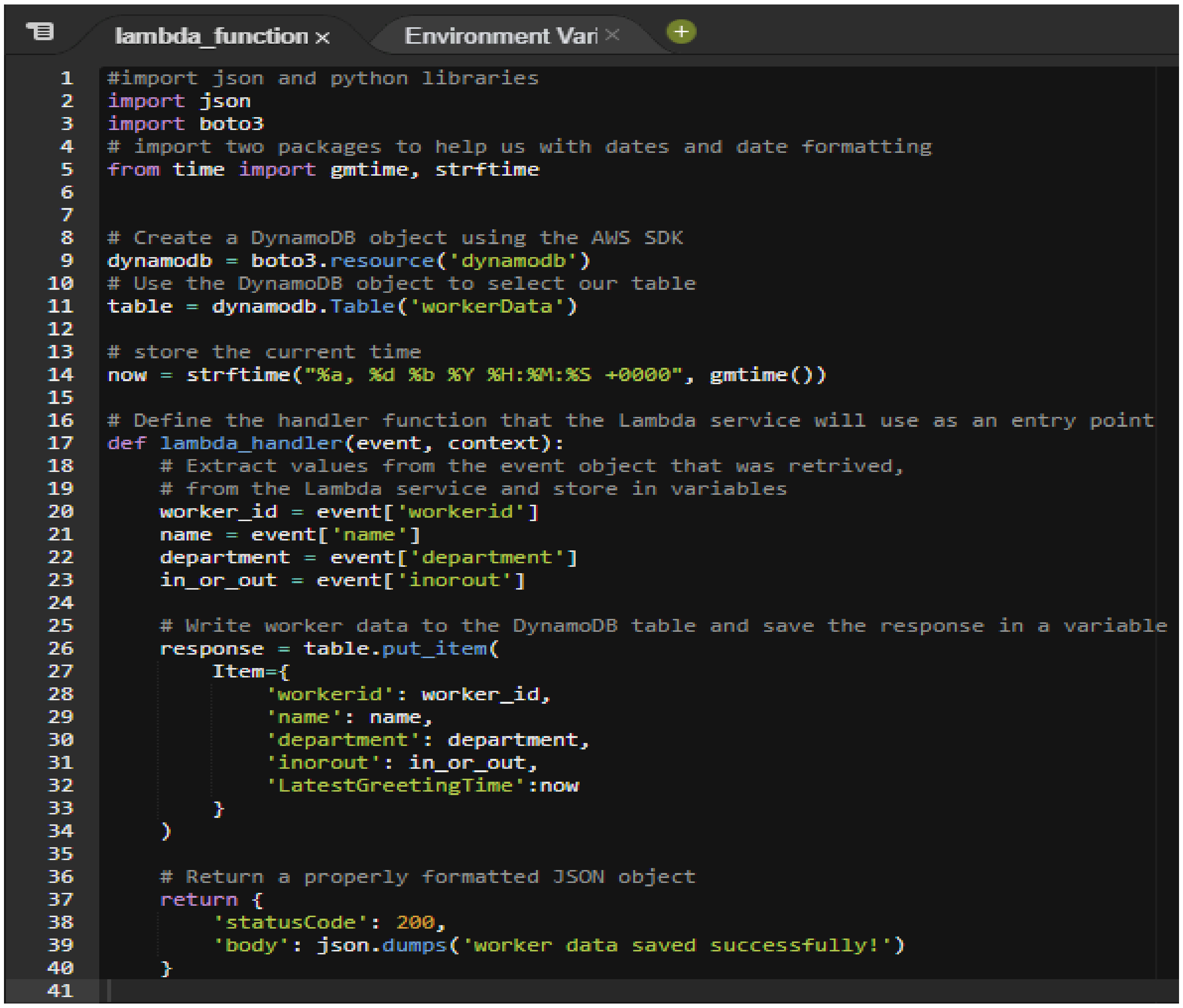
Next came creation of the lambda insert function which would add and update information on the table from user input as shown in figure 26 below the overview of the function, and in figure 27 the code for the insert function.

Figure 26. Lambda insertWorkerData Function overview



Description
-
Last modified
9 days ago
Function ARN
arn:aws:lambda:us-east-1:767398071508:function:insertWorkerData
Function URL
-

Figure 27. Lambda insertWorkerData Function Code

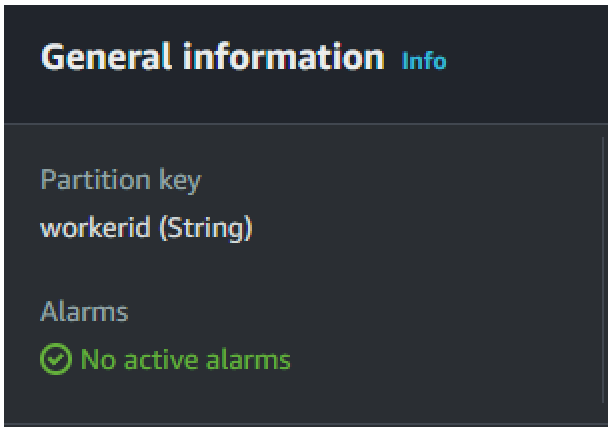


```python
#import json and python libraries
import json
import boto3
# import two packages to help us with dates and date formatting
from time import gmtime, strftime


# Create a DynamoDB object using the AWS SDK
dynamodb = boto3.resource('dynamodb')
# Use the DynamoDB object to select our table
table = dynamodb.Table('workerData')

# store the current time
now = strftime("%a, %d %b %Y %H:%M:%S +0000", gmtime())

# Define the handler function that the Lambda service will use as an entry point
def lambda_handler(event, context):
    # Extract values from the event object that was retrived,
    # from the Lambda service and store in variables
    worker_id = event['workerid']
    name = event['name']
    department = event['department']
    in_or_out = event['inorout']

    # Write worker data to the DynamoDB table and save the response in a variable
    response = table.put_item(
        Item={
            'workerid': worker_id,
            'name': name,
            'department': department,
            'inorout': in_or_out,
            'LatestGreetingTime':now
        }
    )

    # Return a properly formatted JSON object
    return {
        'statusCode': 200,
        'body': json.dumps('worker data saved successfully!')
    }

```

The code is fairly similar to our get Function with the same libraries, time inclusion packages, and same table selections but with a few additions and a missing component that is not needed in a POST function. The component im talking about that's not needed is the region specification since POST works automatically with the region the lambda function is in and cant be applied to anything outside the region the lambda is made in and set to, so it only needs the permission in the IAM Role to communicate, add, and read the tables in DynamoDB meaning you only need to state which table from that region. A new object is made to store and record the current time of the input and we extract the values from the event objects and store it in variables and after words we input the data into the DynamoDb table, and lastly a return response to let us know that everything worked and was successfully saved.

The final tool used to make our whole system work was our DynamoDB, our data storage table tool. Now AWS DynamoDB is a fully managed proprietary NoSQL database that offers a fast persistent key–value datastore with built-in support for replication, autoscaling, encryption at rest, and on-demand backup. Each table is saved to a specified region and can only be communicated within that said region for my system it was us-east-1. It mainly focuses on a set Key input that all things are sorted by and is used to call or scan the table for data, this is the partition key as shown in figure 28 below ours was workerid and it was a string.

Figure 28. DynamoDB table general information

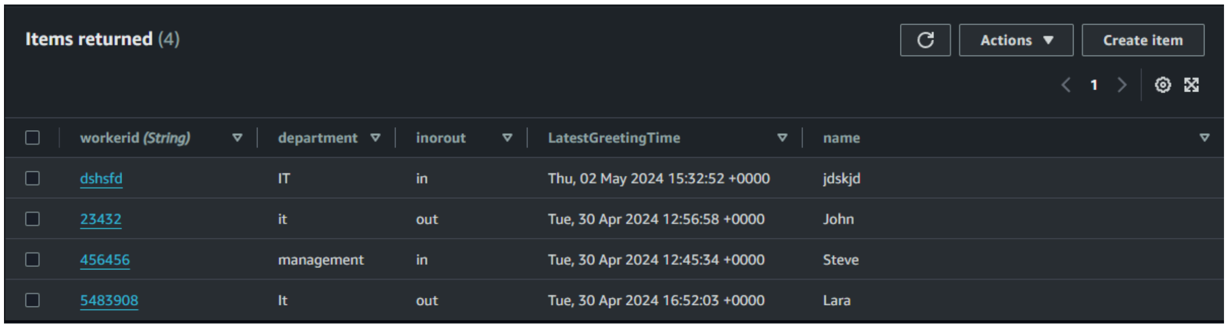

General information Info

Partition key
workerid (String)

Alarms
No active alarms

As for the actual setup of the table it's not too difficult, you just have to decide what type of values you want to use for your data. I went with string since we will be using lowercase, capital case, number, and letters, plus you can use symbols. Also choose worker ID since its the one value that will always be unique to a specific individual and won't repeat instead of names and departments. And an example of our table is shown below in figure 29 on how it looks through the console.

Figure 29. DynamoDB table

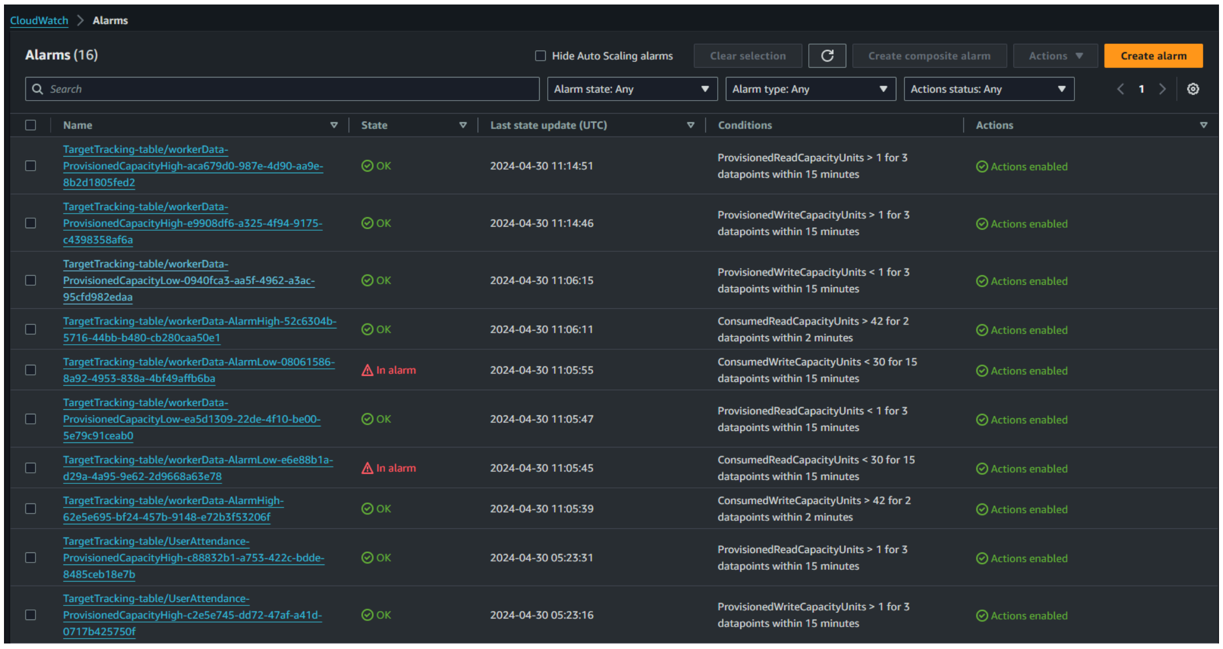


	workerid (String)	department	inorout	LatestGreetingTime	name
☐	dshsfd	IT	in	Thu, 02 May 2024 15:32:52 +0000	jdskjd
☐	23432	it	out	Tue, 30 Apr 2024 12:56:58 +0000	John
☐	456456	management	in	Tue, 30 Apr 2024 12:45:34 +0000	Steve
☐	5483908	It	out	Tue, 30 Apr 2024 16:52:03 +0000	Lara

Only thing left after that was again add the IAM permission so the table can communicate with other tool like all other and testing with my lambda functions if inputs would work and get a positive response since lambda has a built in testing function to run the code with pre built values you basically write a input statement that would be sent and it gets tested.

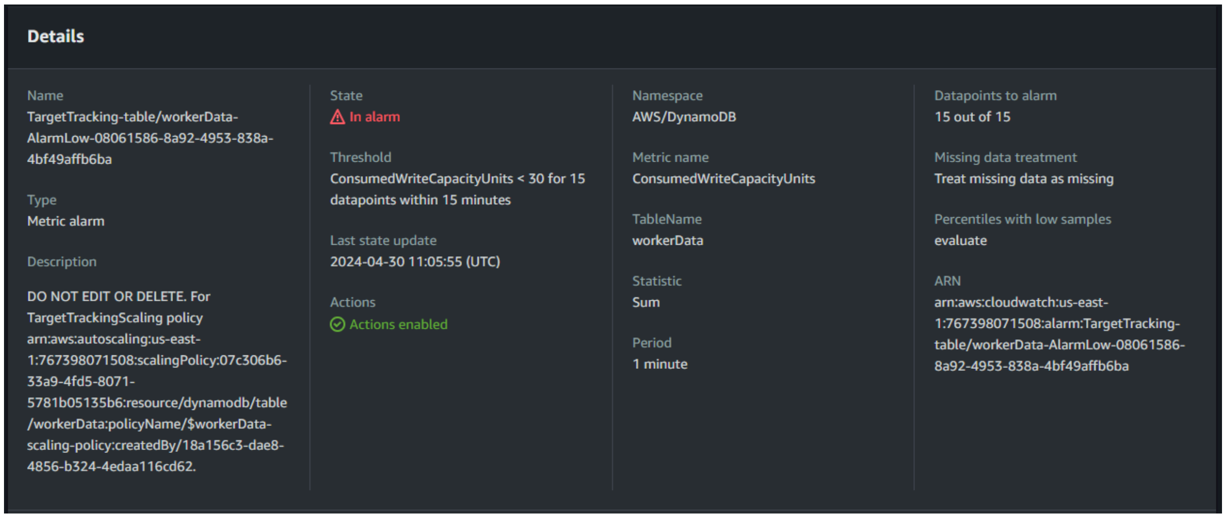
Last tool I would like to mention that was not set up or made by me but used was CloudWatch. CloudWatch is a monitoring service built for DevOps engineers, developers, site reliability engineers (SREs), IT managers, and product owners to keep track of activity and alerts of suspicious activity, errors in code or connections, and pricing changes. I used it mainly to track problems with connectivity and seeing what didn't work, and work backwards from there to correct the problem. An example of this is shown in figure 30 below of the logs, and a more detailed look is given in figure 31.

Figure 30. CloudWatch logs



	Name	State	Last state update (UTC)	Conditions	Actions
☐	TargetTracking-table/workerData-ProvisionedCapacityHigh-aca679d0-987e-4d90-aa9e-8b2d1805fed2	✓ OK	2024-04-30 11:14:51	ProvisionedReadCapacityUnits > 1 for 3 datapoints within 15 minutes	✓ Actions enabled
☐	TargetTracking-table/workerData-ProvisionedCapacityHigh-e9908df6-a325-4f94-9175-c4398358af6a	✓ OK	2024-04-30 11:14:46	ProvisionedWriteCapacityUnits > 1 for 3 datapoints within 15 minutes	✓ Actions enabled
☐	TargetTracking-table/workerData-ProvisionedCapacityLow-0940fca3-aa5f-4962-a3ac-95cfd982edaa	✓ OK	2024-04-30 11:06:15	ProvisionedWriteCapacityUnits < 1 for 3 datapoints within 15 minutes	✓ Actions enabled
☐	TargetTracking-table/workerData-AlarmHigh-52c6304b-5716-44bb-b480-cb280caa50e1	✓ OK	2024-04-30 11:06:11	ConsumedReadCapacityUnits > 42 for 2 datapoints within 2 minutes	✓ Actions enabled
☐	TargetTracking-table/workerData-AlarmLow-08061586-8a92-4953-838a-4bf49affb6ba	⚠ In alarm	2024-04-30 11:05:55	ConsumedWriteCapacityUnits < 30 for 15 datapoints within 15 minutes	✓ Actions enabled
☐	TargetTracking-table/workerData-ProvisionedCapacityLow-ea5d1309-22de-4f10-be00-5e79c91ceab0	✓ OK	2024-04-30 11:05:47	ProvisionedReadCapacityUnits < 1 for 3 datapoints within 15 minutes	✓ Actions enabled
☐	TargetTracking-table/workerData-AlarmLow-e6e88b1a-d29a-4a95-9e62-2d9668a63e78	⚠ In alarm	2024-04-30 11:05:45	ConsumedReadCapacityUnits < 30 for 15 datapoints within 15 minutes	✓ Actions enabled
☐	TargetTracking-table/workerData-AlarmHigh-62e5e695-bf24-457b-9148-e72b3f53206f	✓ OK	2024-04-30 11:05:39	ConsumedWriteCapacityUnits > 42 for 2 datapoints within 2 minutes	✓ Actions enabled
☐	TargetTracking-table/UserAttendance-ProvisionedCapacityHigh-c88832b1-a753-422c-bdde-8485ceb18e7b	✓ OK	2024-04-30 05:23:31	ProvisionedReadCapacityUnits > 1 for 3 datapoints within 15 minutes	✓ Actions enabled
☐	TargetTracking-table/UserAttendance-ProvisionedCapacityHigh-c2e5e745-dd72-47af-a41d-0717b425750f	✓ OK	2024-04-30 05:23:16	ProvisionedWriteCapacityUnits > 1 for 3 datapoints within 15 minutes	✓ Actions enabled

Figure 31. More detailed error report that was clicked on in CloudWatch



Details

Name
TargetTracking-table/workerData-AlarmLow-08061586-8a92-4953-838a-4bf49affb6ba

Type
Metric alarm

Description

DO NOT EDIT OR DELETE. For TargetTrackingScaling policy arn:aws:autoscaling:us-east-1:767398071508:scalingPolicy:07c306b6-33a9-4fd5-8071-5781b05135b6:resource/dynamodb/table/workerData:policyName/$workerData-scaling-policy:createdBy/18a156c3-dae8-4856-b324-4edaa116cd62.

State
⚠ In alarm

Threshold
ConsumedWriteCapacityUnits < 30 for 15 datapoints within 15 minutes

Last state update
2024-04-30 11:05:55 (UTC)

Actions
✓ Actions enabled

Namespace
AWS/DynamoDB

Metric name
ConsumedWriteCapacityUnits

TableName
workerData

Statistic
Sum

Period
1 minute

Datapoints to alarm
15 out of 15

Missing data treatment
Treat missing data as missing

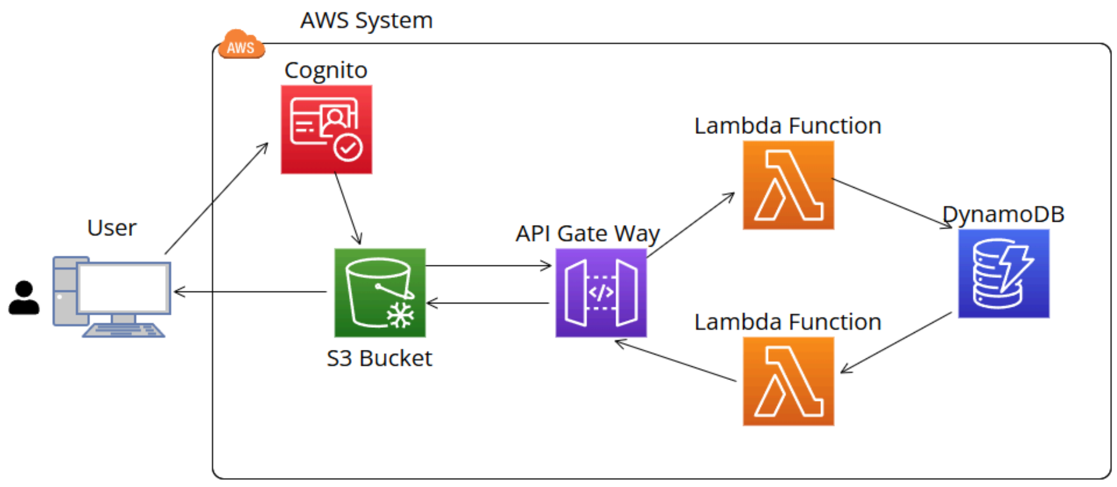
Percentiles with low samples
evaluate

ARN
arn:aws:cloudwatch:us-east-1:767398071508:alarm:TargetTracking-table/workerData-AlarmLow-08061586-8a92-4953-838a-4bf49affb6ba

3.2 Data Analysis

Take a look at Figure 32 below it will remind you of our system set up and what parts are connected.

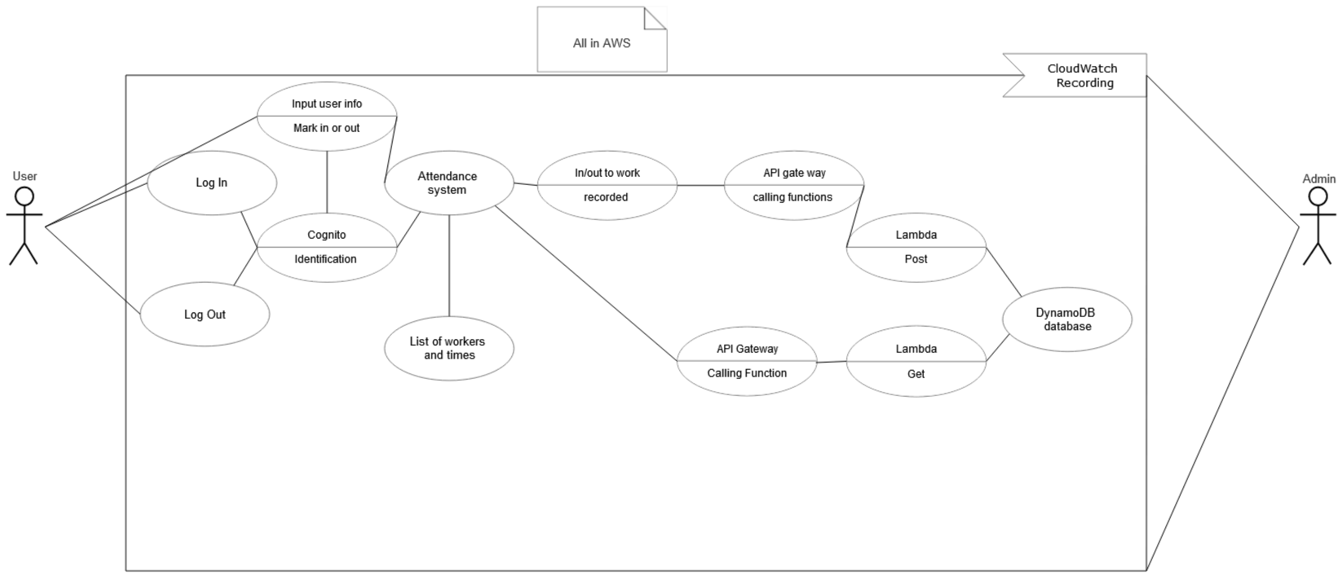
Figure 32. The Attendance system structure and layout



This is our architecture showing how each of the parts are connected in our system

The whole system is located on the cloud which frees up space, processing power, and has added security improving the experience and limiting problems along with making things more convenient. And to give a good demonstration or view of how it all goes and works is a use case diagram to see how it all would work out for someone interaction with the system in figure 33.

Figure 33. Use Case Diagram of Attendance System

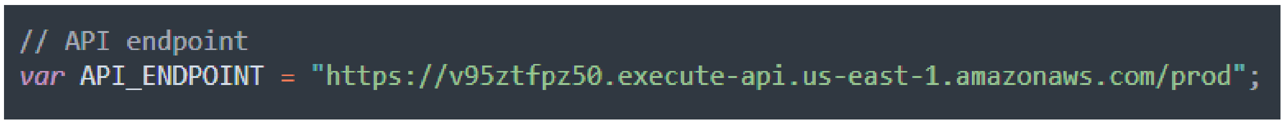


This is a basic rundown of how the interaction would go (i'm not very good at these sort of charts)

The amount of time and actual interaction a user does with the system is very limited and quick which is good. We don't want to overload them or confuse the user, it should be a quick and easy process. It's not the most overly complex system but it gets the job done and delivers the information needed for the task of recording work attendance and if a user is in or not.

A portion I wish to talk about that didn't fit in the architecture is the actual script.js for the website that basically performed the GET and POST actions. As mentioned before in the Architecture portion a API endpoint code was needed for the gateway to accept and transfer data this was done in the script.js the code will be in figure 34 done below.

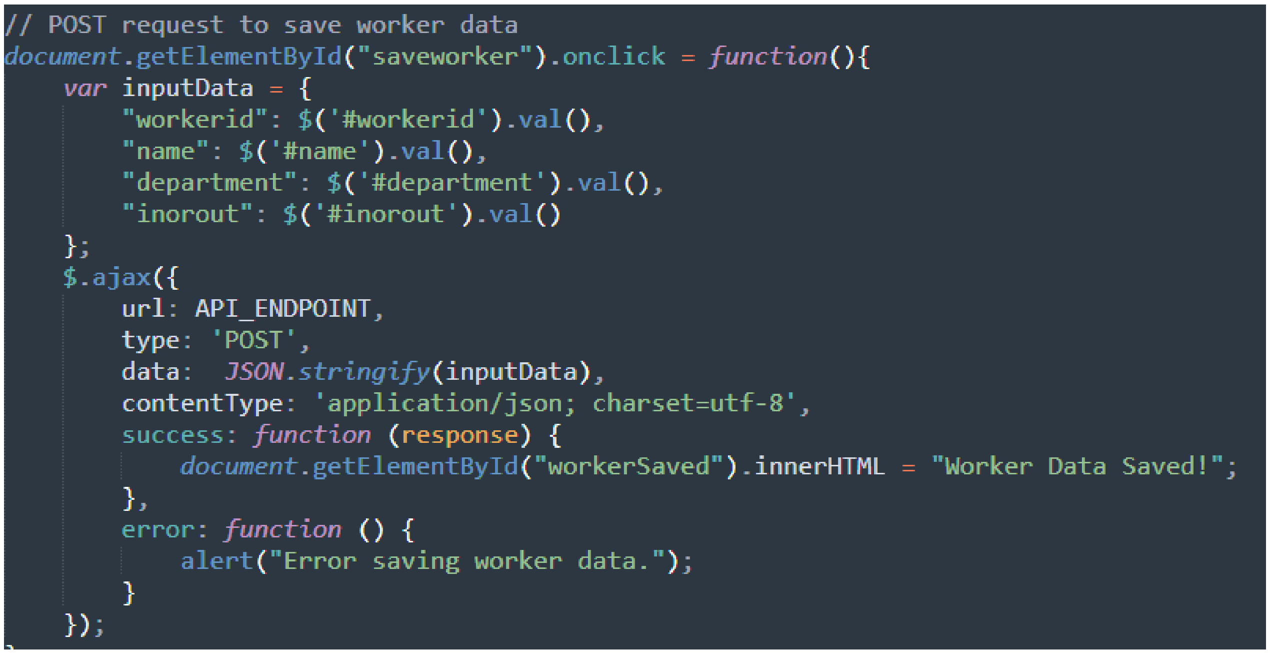
Figure 34. API Endpoint gateway code



```
// API endpoint
var API_ENDPOINT = "https://v95ztfpz50.execute-api.us-east-1.amazonaws.com/prod";
```

Now how the code is used we need to take a look at figure 35 which shows the POST function portion of the code.

Figure 35. POST code in scripts.js

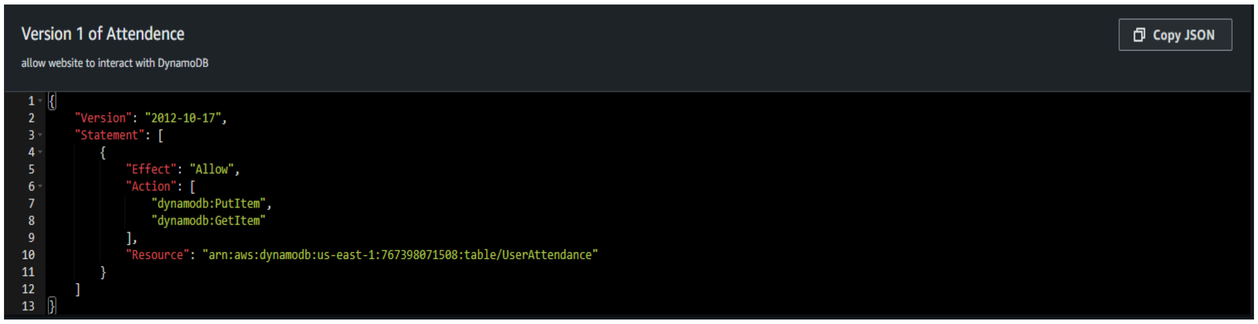


```
// POST request to save worker data
document.getElementById("saveworker").onclick = function(){
    var inputData = {
        "workerid": $('#workerid').val(),
        "name": $('#name').val(),
        "department": $('#department').val(),
        "inorout": $('#inorout').val()
    };
    $.ajax({
        url: API_ENDPOINT,
        type: 'POST',
        data:  JSON.stringify(inputData),
        contentType: 'application/json; charset=utf-8',
        success: function (response) {
            document.getElementById("workerSaved").innerHTML = "Worker Data Saved!";
        },
        error: function () {
            alert("Error saving worker data.");
        }
    });
```

Here you see in the code the url portion that includes the code or calls it from figure 34 so that the API gateway lets it through and knows to enact the Lambda functions. And once the API gateway is called and does an action this triggers the Lambda to grab the data from the website that was attempted to submit, bringing it to the database. And before all this it checks if you have the proper IAM credentials because if you don't the gateway will stop you and not let anything through. The GET request also uses the url call to work and retrieve the data from the table.

Lastly, learning to understand and code policies and the formatting for them turned out simple but can be complex due to understanding what access and permission are needed. As shown in figure 36 this is simple and minimal but understanding what you give access for and to is the challenge and figuring out the pathing to specific tools and databases can be annoying.

Figure 36. Policy coding and writing



Version 1 of Attendence

allow website to interact with DynamoDB

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:PutItem",
                "dynamodb:GetItem"
            ],
            "Resource": "arn:aws:dynamodb:us-east-1:767398071508:table/UserAttendance"
        }
    ]
}
```

The pathing being the resources section.

4. Challenges and Takeaways

The first challenge that I got or faced was just time management, and getting everything done myself since I had other projects and classes going on I had to focus on those at times. Being alone proves its own challenges and benefits mainly being I don't have to worry about someone not doing anything and I always know the state of the project. But also I'm alone. I have to research everything, I can't really ask for help when something goes wrong. I can't divide up work for all users to get this done timely and orderly with no stress.This provides a good learning experience why teams are made and used in the workforce more often than a single individual which was a good take away from all this.

The biggest challenge and takeaway I feel is that AWS is extremely secure which is a good thing in safety, but a hard grueling lesson when coding and connecting everything together into a usable system. I was previously taught you need the IAM credentials for you to do everything and make stuff work which is true in a sense. What no one else really bothers to teach and tell you is how you need to link everything to IAM rolls and giver individual permission to tools in AWS along with opening up certain pathways and configuring settings. I had the proper IAM credentials so many times and just tried to get things to run or work and they wouldn't. I would be blocked by an error, and I would of course start searching for why it does this and everyone just goes do you have the right IAM credentials on your account. Finding out that I have to write out custom Policies and link them to the services I'm using was a huge breakthrough and clarification for me once I found out. The amount of times I ran my Lambda functions in testing environment to just get no connection or couldn't connect was infuriating and i kept checking my coed to see if I did something wrong but no it was good just didn't see that I had to make a policy and link it to Lambda for it to talk to DynamoDB. This was a good learning lesson and really showed me the complexity of AWS and the many components needed to make it work.

Along with this came all the other hidden systems and features in AWS that they don't really tell you up front unless you go looking such as CORS, Domains, AWS STC, DNS record indication, ect. There were some security measures and features I had to tinker with and edit along with figuring out how they work to connect things and make sure I had access. Really showed the importance of cyber security and how it can affect things along with keeping AWS safe and reliable.

The next real challenge and take away was API gateway itself. At first I didn't even know about it for some reason i thought that my website since it was on AWS could just talk to the lambda functions and grab the data and input it with the right script but nothing would connect,and of course when I went to look up solutions first thing ever was do you have the right permission from IAM and if my code was right. It's not until later when looking through some AWS examples on there guide page and how to do things like websites and such did they mention connecting the systems though API gateway and explained that the static site I made on the bucket can not communicate with the AWS system in general its considered outside the system so you have to link it with the gateway to call functions. This was one of the major problems and take-aways I got out of doing this project.

Lastly was just facing reality and knowing limits or what is capable of being done there so many features and ideas I had for this system that just wouldn't work in my budget and time frame with the limited number of work I can do this will be further talked about in the next section. It made me look at things more realistically and temper expectations to what realistically can be done and what is capable in your set scope of time and resources.

5. Future

As I stated previously there were a couple of features and parts to the project that could not be included for reasons such as time, and money, or I couldn't figure it out fully and had to drop it. The first feature dropped in early development was facial recognition software to identify the user to login to attendance. I could get the software to learn and identify faces but when it came to connect everything on AWS it just didn't want to work or identify faces that were saved in this database, so I can check this one done but I just couldn't figure it out. But it could be a great feature to add and speed up the login and sign in features wasting less time even if it could be considered less secure. The next feature was just doing identity cards to swipe or scan that you're in or out of work. The main reason for this one being dropped was just cost, mainly since it didn't fit into budget and I would have to buy the machines and set it up. This feature I feel is a bit safer then users and passwords but can fall short because of the human aspect of being irresponsible with either losing it or damaging it. Originally I would have it require you to scan a card and do a facial recognition test to login for security reasons but as stated you can see why it was dropped. Another big feature not included was going to be a second database that would record in and out separately. This just came down to cost and some technical errors of things just responding properly to my code so mainly I just couldn't figure it out and it would cost a bit to have the second one up and running. Outside of all that I would make some improvement to UI and some extra security safety measures I could implement with API gateway but this came down to me running out of time and having so much trouble fixing those errors I kept getting.

6. Additional Notes

There's not much I could say here other than I feel like my report comes out a bit short maybe from the specified numbers given and I would just be repeating myself if I go into more details since there's only so much you can write. Maybe it's since I'm alone or I just couldn't think of anything more important to write about but I feel it explains the project well and gives an idea of AWS and its system to all.

7. References

[1] Using cross-origin resource sharing (CORS) - Amazon Simple Storage Service. Available from: https://docs.aws.amazon.com/AmazonS3/latest/userguide/cors.html,

[2] Cloud computing services - Amazon Web Services (AWS). Amazon Web Services, Inc. Available from: https://aws.amazon.com/,

[3] Identity and Access Control - Introduction to AWS Security. Available from: https://docs.aws.amazon.com/whitepapers/latest/introduction-aws-security/identity-and-access-control.html,

[4] Welcome - Amazon Cognito user pools. Available from: https://docs.aws.amazon.com/cognito-user-identity-pools/latest/APIReference/Welcome.html,

[5] Policies and permissions in IAM - AWS Identity and Access Management. Available from: https://docs.aws.amazon.com/IAM/latest/UserGuide/access_policies.html,

[6] Buckets overview - Amazon Simple Storage Service. Available from: https://docs.aws.amazon.com/AmazonS3/latest/userguide/UsingBucket.html,

[7] What is Amazon API Gateway? - Amazon API Gateway. Available from: https://docs.aws.amazon.com/apigateway/latest/developerguide/welcome.html,

[8] Introduction to AWS Lambda &amp; serverless applications (56:01). Amazon Web Services, Inc.

[9] Setting up DynamoDB - Amazon DynamoDB. Available from: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/SettingUp.html,

[10] How Amazon CloudWatch works - Amazon CloudWatch. Available from: https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/cloudwatch_architecture.html,