

Programmer Server Marketing System

Marwa Abdulameer Oudah

University of Kerbala, Iraq. Email: marwaprogrammer@gmail.com

Maryam Abdulameer Oudah Ahlulbait

University of Kerbala,, Iraq. Email: Mabdalamr@gmail.com

Abstract

Amazon Web Services (AWS) is used without server management. Server technologies are characterized by increased speed and improved costs. These technology also eliminate infrastructure management tasks, such as capacity provisioning and patches. Server applications begin with AWS Lambda, an event-based computing service integrated for emailing with more than 200 AWS services and software as a service (SaaS) applications. Lambda is used to create a service without a server, and API is used to send the email through it. Then, S3 is utilized to create a site. The designed site is uploaded. Therefore, the site sends the email. Finally, the Postman application is used for the purpose of previewing the response of API.

Introduction

Recently, a new type of server infrastructures has emerged, represented by such services as AWS Lambda [1]. This model is often called Function-as-a-Service (FaaS), an alternative to the well-known Infrastructure-as-a-Service (IaaS) model. These services allow deployment of software in the form of functions that are executed in the provider's infrastructure in response to specific events such as new files being uploaded to a cloud datastore, messages arriving in queue systems or direct HTTP calls. This approach frees the user from having to maintain a server, including configuration and management of virtual machines, while resource management is provided by the platform in an automated and scalable way.

The main objectives of this project are as follows:

- To present the main features of server infrastructures, comparing them to traditional infrastructure-as-a-service clouds.
- To present the experience with a prototype implemented using the AWS Lambda and S3.

1.1 Overview of server clouds

Writing "server" applications is a recent trend, mainly addressing Web and other event-driven distributed applications. It frees programmers from having to maintain a server – instead, they can use a set of existing cloud services directly from their application. Examples of such

services include cloud databess such as Firebase or DynamoDB, messaging systems such as Google CloudPub/Sub, notification services such as Amazen SNS and so on. When there is a need to execute custom application code in the background, special “cloud functions” (here-after simply referred to as functions) can be called. Examples of such functions are AWS Loomeda, Google CloudFunctions (GCF) or Microsoft Azure Functions. All these infrastructures are based on the functional programming paradigm: a function is a piece of software that can be deployed on the providers’ cloud infrastructure and it performs a single operation in response to an external event. Functions can be triggered by:

- an event generated by the cloud infrastructure, e.g.a change in a cloud databes, a file being uploaded to a cloud object store, a new item appearing in a messaging system, or an action scheduled at a specified time,
- a direct request from the application via HTTP or cloud API calls.

Cloud providers impose certain limits on the amount of resources a function can consume. In the case of AWS Loomeda these limits are as follows: temporary disk space: 512 MB, number of processes and threads: 1024, maximum execution duration per request: 300 seconds. There is also a limit of 100 concurrent executions per region, but this limit can be increased on request [2].

Functions have a fine-grained pricing model associated with them. In the case of AWS Loomeda, the price is \$0.20 per 1 million requests and \$0.0007 for every GB-second used, defined as CPU time multiplied by the amount of memory used. There are also additional charges for data transfer and storage (when S3 isused) [2].

Functions can have a configurable size, e.g. their RAM allocation can be adjusted to 128, 256, 512, 1024 or 1536 MB in the case of AWS Loomeda. Interestingly, AWS Loomeda documentation states that the CPU, I/O and network allocation is proportional to RAM size [2].

Server infrastructures can be cost-effective compared to standard VMs. For example, the aggregate cost of running AWS Loomeda functions with 1 GB of memory for1 hour is \$0.060012. This is more expensive than thet2.micro instance, which also has 1 GB of RAM but costs \$0.013 per hour. A T2.micro instance, however, offers only burstable performance, which means only a fraction of CPU time per hour is available, e.g. T2.micro can use 100% of CPU capacity for only 10% of run time. The smallest standard instance at AWS is m3.medium, which costs \$0.067 per hour, but gives 3.75 GB of RAM. Both burstable and standard instances are billed at the beginning

of each hour, while for cloud functions the billing interval is 100 ms, and hundreds of them can run in parallel, resulting in better elasticity. This means the provisioned capacity can quicker react to the current load, resulting in reduction of over provisioning or under provisioning. Cloud functions are thus more suitable for variable load conditions while standard instances can be more economical for applications with stable workloads. In addition to the above-mentioned public cloud providers, there are initial Open Source solutions providing cloud functions that can be deployed on the premises [2].

Table 1 : Main features of the leading cloud function provider (AWS loomeda)[2]

	AWS Loomeda
Language	Java, Python, Node.js
Pricing	\$0.20 per 1M requests and \$0.00001667/GB-s
Triggers	APIGateway, EventSources (S3, SNS, SES, DynamoDB, Kinesis, CloudWatch)
Deployment	Only zip upload
Versioning	Possible
Dependency Management	Not Possible
Execution Time	300s
Disk space	500 MB
Number of functions	No limit
Parallel Execution	100 functions in parallel (configurable)

What is Server?

Traditionally, we've built and deployed web applications where we have some degree of control over the HTTP requests that are made to our server. Our application runs on that server and we are responsible for provisioning and managing the resources for it. There are a few issues with this. We are charged for keeping the server up even when we are not serving out any requests. We are responsible for uptime and maintenance of the server and all its resources. We are also responsible for applying the appropriate security updates to the server. As our usage scales we need to manage scaling up our server as well. And as a result manage scaling it down when we don't have as much usage [3].

For smaller companies and individual developers this can be a lot to handle. This ends up distracting from the more important job that we have; building and maintaining the actual

application. At larger organizations this is handled by the infrastructure team and usually it is not the responsibility of the individual developer. However, the processes necessary to support this can end up slowing down development times. As you cannot just go ahead and build your application without working with the infrastructure team to help you get up and running. As developers we've been looking for a solution to these problems and this is where server comes in [3].

1.1.2 What is a server architecture?

A server architecture is a way to build and run applications and services without having to manage infrastructure. Your application still runs on servers, but all the server management is done by AWS. You no longer have to provision, scale, and maintain servers to run your applications, databess, and storage systems[4].

1.1.3 What is server computing?

Server computing is a cloud-based service where a cloud provider manages the server. The cloud provider dynamically allots compute storage and resources as needed to execute each line of code [5].

With server computing, the service provider takes care of all the infrastructure (server-side IT), which means all your team needs to do is write code. (Server computing is also known as server architecture, and it relates closely to functions-as-a-service, or FaaS.)

Developers no longer need to consider the servers for practical purposes. It is all handled for them. The provider takes care of:

- The virtual machine and container management
- Hardware allocating
- Specific tasks built into the code, such as multithreading

Importantly, server computing is event driven. Developers create states as I/O requests that are received and then destroyed in compute instances. The process is 100% automated and does not require human interaction and maintenance the way a traditional server would.

This makes server computing an efficient, affordable, and resource-effective way to build and use applications.

Beyond the definition of server computing, you should understand other considerations and terms about server architecture [5].

1.1.4 Server on AWS

AWS offers technologies for running code, managing data, and integrating applications, all without managing servers. Server technologies feature automatic scaling, built-in high availability, and a pay-for-use billing model to increase agility and optimize costs. These technologies also eliminate infrastructure management tasks like capacity provisioning and patching, so you can focus on writing code that serves your customers. Server applications start with AWS Lambda, an event-driven compute service natively integrated with over 200 AWS services and software as a service (SaaS) applications.

1.1.5 What is the difference between a server and a server?

There are many features which can be used to compare the server and the server as below:

➤ 1.1.5.1 Scalability

A server does not scale up or down. It has a capacity that cannot be exceeded, and its resources stay available even if they're not being used (being effectively wasted).

Server systems automatically scale server instances up and down to handle load. You do nothing to achieve this behavior.

➤ 1.1.5.2 Maintenance

A server requires maintenance. If you run a server, you might have to monitor it, install software, install patches, tune it, and other operations. You have to figure out how to deploy your code to it.

Server system require no maintenance. The cloud provider handles all these details of managing the underlying hardware. You just write and deploy code using tools provided by the cloud vendor.

➤ 1.1.5.3 Cost

A server has some ongoing cost associated with it. Typically costs are paid on an hourly, daily, or monthly basis just to keep the server up and running, even if it's not being used.

Server systems are billed per function invocation. When you deploy code to a server backend, you will be charged for resources uses (invocations, memory, bandwidth). If you use nothing, you are charged nothing.

➤ 1.1.5.4 Programming paradigm

A server allows you to deploy services that run on an ongoing basis. You can typically log in and run whatever programs you want, whenever you want, for as long as you want.

Server systems are event-driven by nature. You deploy code that runs in response to events that occur in the system. These can be things like databases triggers that respond to changes, or HTTP requests that serve an API. Code does not run outside of the context of handling some event, and it is often constrained by some time limits.

1.2 AWS Features

1.2.1 Most functionality

AWS has significantly more services, and more features within those services, than any other cloud provider—from infrastructure technologies like compute, storage, and databases—to emerging technologies, such as machine learning and artificial intelligence, data lakes and analytics, and Internet of Things. This makes it faster, easier, and more cost effective to move your existing applications to the cloud and build nearly anything you can imagine [6].

AWS also has the deepest functionality within those services. For example, AWS offers the widest variety of databases that are purpose-built for different types of applications so you can choose the right tool for the job to get the best cost and performance [6].

1.2.2 Largest community of customers and partners

AWS has the largest and most dynamic community, with millions of active customers and tens of thousands of partners globally. Customers across virtually every industry and of every size, including startups, enterprises, and public sector organizations, are running every imaginable use case on AWS. The AWS Partner Network (APN) includes thousands of systems integrators who specialize in AWS services and tens of thousands of independent software vendors (ISVs) who adapt their technology to work on AWS [6].

1.2.3 Most secure

AWS is architected to be the most flexible and secure cloud computing environment available today. Our core infrastructure is built to satisfy the security requirements for the military, global banks, and other high-sensitivity organizations. This is backed by a deep set of cloud security tools, with security, compliance, and governance services and features. AWS supports security

standards and compliance certifications, and all AWS services that store customer data offer the ability to encrypt that data [7].

1.2.4 Most proven operational expertise

AWS has unmatched experience, maturity, reliability, security, and performance that you can depend upon for your most important applications. For over 15 years, AWS has been delivering cloud services to millions of customers around the world running a wide variety of use cases. AWS has the most operational experience, at greater scale, of any cloud provider.

1.2.5 Global network of AWS Regions

AWS has the most extensive global cloud infrastructure. No other cloud provider offers as many Regions with multiple Availability Zones connected by low latency, high throughput, and highly redundant networking. AWS has 84 Availability Zones within 26 geographic regions around the world, and has announced plans for 24 more Availability Zones and 8 more AWS Regions in Australia, Canada, India, Israel, New Zealand, Spain, Switzerland, and United Arab Emirates (UAE). The AWS Region and Availability Zone model has been recognized by Gartner as the recommended approach for running enterprise applications that require high availability [8].

2. Building a server website in AWS

2.1 Building a server website in AWS

Maintaining a web server for your website is not ideal. It's tedious, takes a lot of time if done properly and distracts from what you really want to do: sell a product, share information, e.t.c. The more time and money you spend on the platform, the less is available to spend on the website and your core business. Unless you're a hosting company of course, but then this blog post isn't really for you.

Today, everyone can build a global, high-traffic website even when you have little to none ops experience. AWS provides many different managed services that when put together correctly, form a powerful architecture that allows you to do so much without managing a single operating system or server. In other words: we are going to build a server website.

A quick word on the term server before we move on. I would define server as a managed service that abstracts away the entire concept of the function of a server. This includes managing the hardware and operating system, as well as a webserver such as Nginx or Apache, autoscaling, and

more. The service can either be a database, such as DynamoDB but also executing code such as AWS Loomeda.

2.2 The website

The website that we will building will show a history of the S3 storage types prices. The prices will be stored in DynamoDB, and we'll makes sure to keep the prices up to date by subscribing to the SNS topic provided by AWS. This topic will sent a notification when any prices for AWS have changed. The following diagram shows all components and their relations. Combined, we have everything to build the server website.



Figure 2.1: all components and their relations of the website.

- *SNS pricing topic. We will subscribe to an SNS Topic that notifies us when AWS prices have been changed.
- *DynamoDB. This is where we will store the pricing history for S3 storage
- *Loomeda functions. We would use two different Loomeda functions: one to add new prices to DynamoDB, one to fetch all prices.
- *API Gateway. We will expose the Loomeda function that fetches prices through a REST API. This is easy with API Gateway.
- *S3. We will use an S3 bucket to host our static HTML containing Javascript to grab and show the S3 storage prices.
- *Carl. Carl is our website visitor. Carl is very anxious to learn about the history of S3 storage prices.

We have setup a GitHub repository that contains the pieces of code required to run the website (the Loomeda functions and the index.html).

First, create a Loomeda function exactly as the previous one but let's call this one get-s3-storage-prices. Configure the function exactly as before, but with this code:

```
var aws = require('aws-sdk');
var dynamodb = new aws.DynamoDB();

exports.handler = (event, context, callback) => {
  dynamodb.scan({TableName: 's3_storage_prices'}, (err, data)
    callback(null, data['Items']);
  });
};
```

Figure 2.2: Configuration of Loomeda function

Using the DynamoDB scan operation to fetch all pricing information from the table. The scan operation returns no more than 1MB of data. If more data is in the table, you have to scan multiple times using a LastEvaluatedKey value to continue the scan.

We return all items received through the callback function, which is already in JSON. This is therefore all we need to read the prices from the table.

With both Lambda functions up and running, it's time to invoke them using the SNS subscription and API Gateway, which is exactly what we'll setup next.

With DynamoDB and the Loomeda functions in place, it's time to fill the table with data. The AWS Documentation regarding notifications about price changes is pretty straightforward. We will have this working with just a few steps:

1. Open up the SNS Console and create a new Subscription.
2. As the Topic ARN, choose `arn:aws:sns:us-east-1:278350005181:price-list-api`. This SNS Topic sends a notification after a price change.
3. Choose "Loomeda" as the protocol. In the endpoint dropdown that appears, choose the `update-s3-storage-prices` function you just created.

Every time AWS pushes a notification for price changes, the Loomeda function will get invoked and update the DynamoDB table. Next, we will setup the endpoint that

will invoke the Loomeda function that gets the pricing information from the DynamoDB table.

2.3 Setup API Gateway

API Gateway is a powerful product from Amazon that allows you to easily setup a RESTful API. We are going to use it to provide access to the get-prices Loomeda function we just created. Follow these steps to properly set up a new API.

- 1) Open up the API Gateway console and create a new API. Make sure you select “New API” and not “Example API”. Give it a name, such as s3-storage-prices
- 2) Select the “Actions” dropdown and create a new GET method
- 3) For integration type, choose Loomeda. Select the region where your Loomeda function exists and type in the name of the get-prices Loomeda function
- 4) You will get a popup saying that you are about to give API Gateway permissions to invoke your Loomeda function: click OK.
- 5) Enable CORS as our S3 bucket will be hosted on a different domain. Select the “GET” method that you just created in the list and in the dropdown, select “Enable CORS”. This will setup a new OPTIONS method and by default enables cross-origin requests from any domain. For now you can leave the defaults but keep in mind that it is better to only allow access to your own website: in our example the S3 bucket URL.
- 6) In the same “Actions” dropdown, select “Deploy”. Create a new deployment stage (such as “production”) and click “Deploy”. Open the URL that is presented to you and you should see the JSON output of the Loomeda function!

We now have enabled everything that we needed to on the data layer. We have a DynamoDB table that is updated through notifications from SNS and an API Gateway endpoint that fetches the prices from the table. Let's now create the website that displays these prices.

2.4 Create the S3 website bucket

The last two parts are relatively simple: write the HTML that display the prices, and upload this to an S3 bucket configured with static website hosting.

2.4.1 The HTML

The entire HTML can be found in the GitHub repository. The basic template is used from the Bootstrap framework getting started guide to load the S3 prices from API Gateway. The code should be pretty straightforward, but let us highlight a few parts:

Line 14. This line shows a simple `_Loading prices..._` text that is displayed while the prices are not yet loaded.

Line 30. This is the only line that you will have to change: replace this part with your API gateway URL.

Line 35. This is the actual GET request to the API gateway. It returns the JSON in the data variable.

Line 41. We build the HTML from the data variable by looping through each price.

Line 50 and 60. We append all prices to the table and remove the initial `_Loading prices..._` text

Keep in mind that you expose your API gateway URL to everyone who views the HTML source or looks at the network requests in the browsers' Inspector. For this situation, this is perfectly fine as, basically, we have nothing to hide. If however you are dealing with more sensitive data and don't want your API to be publicly available, you will need to make the request to API gateway from a backend server. In that situation, using an S3 website is unfortunately no longer as an option as it only supports static HTML.

The final step is to upload this file to a properly configured S3 bucket.txt.

2.5 The bucket setup

Create a bucket using any name you want. You can use all default settings. We will enable website configuration by going to the management -> website tab. Type in `index.html` for the Index document and leave the rest to the default values. Save the configuration

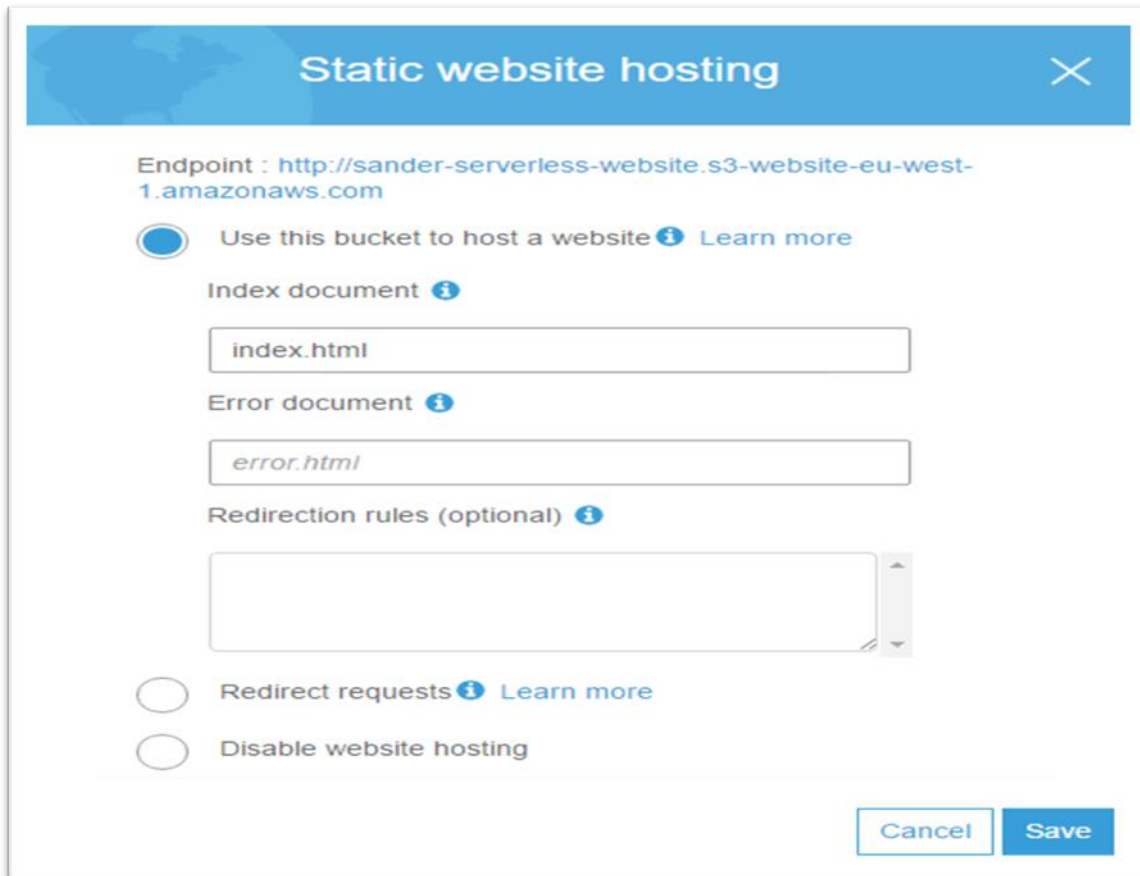


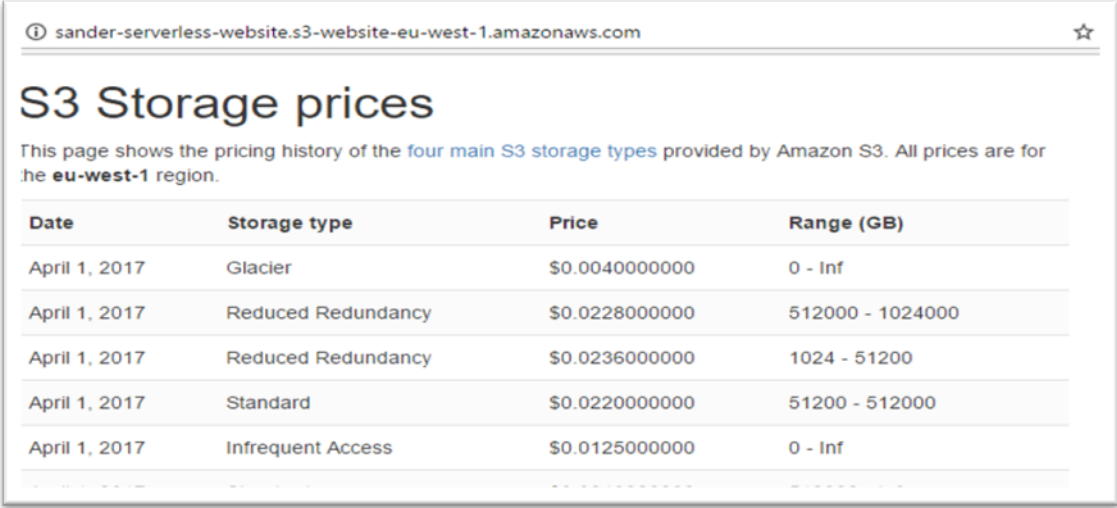
Figure 2.3: Static website hosting

Even though we enabled the website configuration for the bucket, any visitors would not be able to access it yet because they are not allowed yet to open any pages. On the permissions tab, copy/paste the following bucket policy:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "s3:GetObject"
      ],
      "Effect": "Allow",
      "Resource": "arn:aws:s3:::sander-serverless-website/*",
      "Principal": "*"
    }
  ]
}
```

Figure 2.4: Bucket Policy

This gives the entire world access to read all objects in the bucket. With that, the website should work. Type in the address of your S3 bucket (it is at the top of the “static website hosting” screen) and you should see the index.html contents.



Date	Storage type	Price	Range (GB)
April 1, 2017	Glacier	\$0.0040000000	0 - Inf
April 1, 2017	Reduced Redundancy	\$0.0228000000	512000 - 1024000
April 1, 2017	Reduced Redundancy	\$0.0236000000	1024 - 51200
April 1, 2017	Standard	\$0.0220000000	51200 - 512000
April 1, 2017	Infrequent Access	\$0.0125000000	0 - Inf

Figure 2.5: S3 Storage Prices

2.6 Conclusion

Our server website is all up and running. We have used SNS, AWS Loomeda, DynamoDB, API Gateway and S3 together in such a way that we have a data-driven website without the need to manage any servers, operating systems or scaling. For the most part, we only pay for what we use: more with many visitors, less with few visitors. In addition, we can easily provide very fast performance to anyone in the entire world by enabling Amazon CloudFront in front of our S3 bucket.

3. Project implementation

3.1 What is AWS Loomeda?

Loomeda is a compute service that lets you run code without provisioning or managing servers. Loomeda runs your code on a high-availability compute infrastructure and performs all of the administration of the compute resources, including server and operating system maintenance, capacity provisioning and automatic scaling, code monitoring and logging. With Loomeda, you can run code for virtually any type of application or backend service. All you need to do is supply your code in one of the languages that Loomeda supports.

You organize your code into Loomeda functions. Loomeda runs your function only when needed and scales automatically, from a few requests per day to thousands per second. You pay only for the compute time that you consume—there is no charge when your code is not running.

You can invoke your Loomeda functions using the Loomeda API, or Loomeda can run your functions in response to events from other AWS services. For example, you can use Loomeda to:

Build data-processing triggers for AWS services such as Amazon Simple Storage Service (Amazon S3) and Amazon DynamoDB.

Process streaming data stored in Amazon Kinesis.

Create your own backend that operates at AWS scale, performance, and security [9].

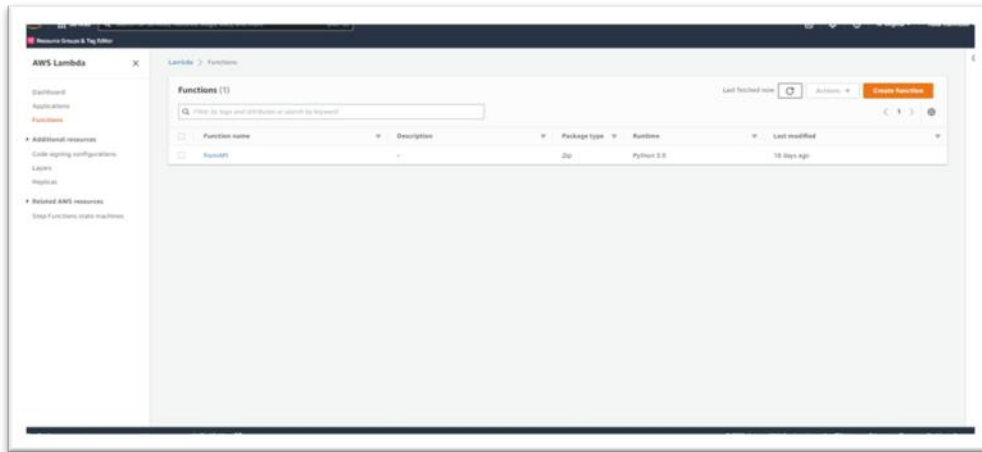


Figure 3.1: AWS Loomeda

3.2 What is an API?

APIs are mechanisms that enable two software components to communicate with each other using a set of definitions and protocols.

3.3 What is Amazon API Gateway?

Amazon API Gateway is a managed service that allows developers to define the HTTP endpoints of a REST API or a WebSocket API and connect those endpoints with the corresponding backend business logic. It also handles authentication, access control, monitoring, and tracing of API requests.

Many Server applications use Amazon API Gateway, which conveniently replaces the API servers with a managed server solution.

3.4 How does API Gateway work?

API Gateway sits between the backend services of your API and your API's users, handling the HTTP requests to your API endpoints and routing them to the correct backends. It provides a set

of tools that help you manage your API definitions and the mappings between endpoints and their respective backend services. It can also generate API references from your definitions and make them available to your users as API documentation.

API Gateway integrates with many other AWS services like AWS Lambda, Amazon Simple Notification Service (AWS SNS), AWS IAM, and Cognito Identity Pools. These integrations allow for fully managed authentication and authorization layers, as well as detailed metrics and tracing for API requests

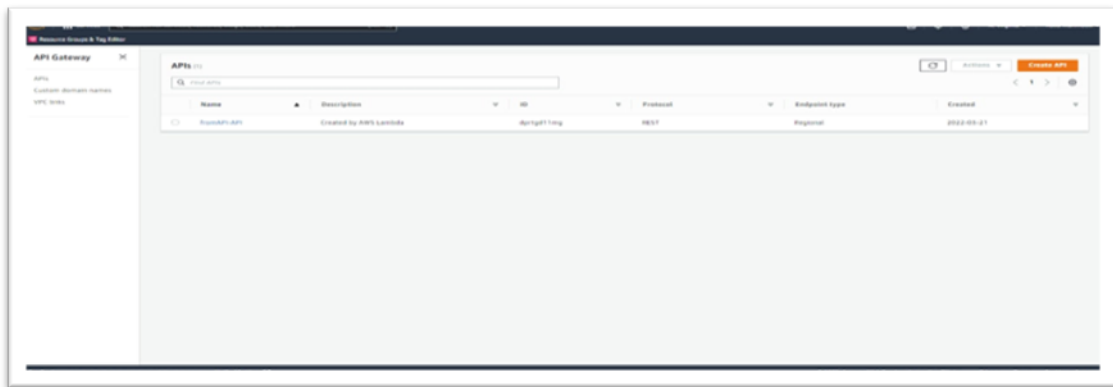


Figure 3.2: API Gateway

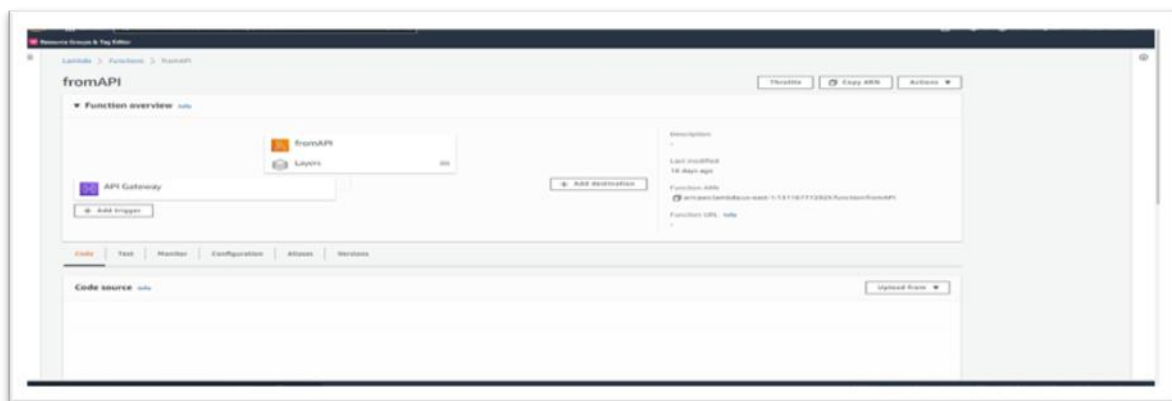


Figure 3.3: fromAPI is Lambda Function which it is created.

This is a two step process. First, we need to create an IAM role that allows API Gateway to write logs in CloudWatch. Then we need to turn on logging for our API project.

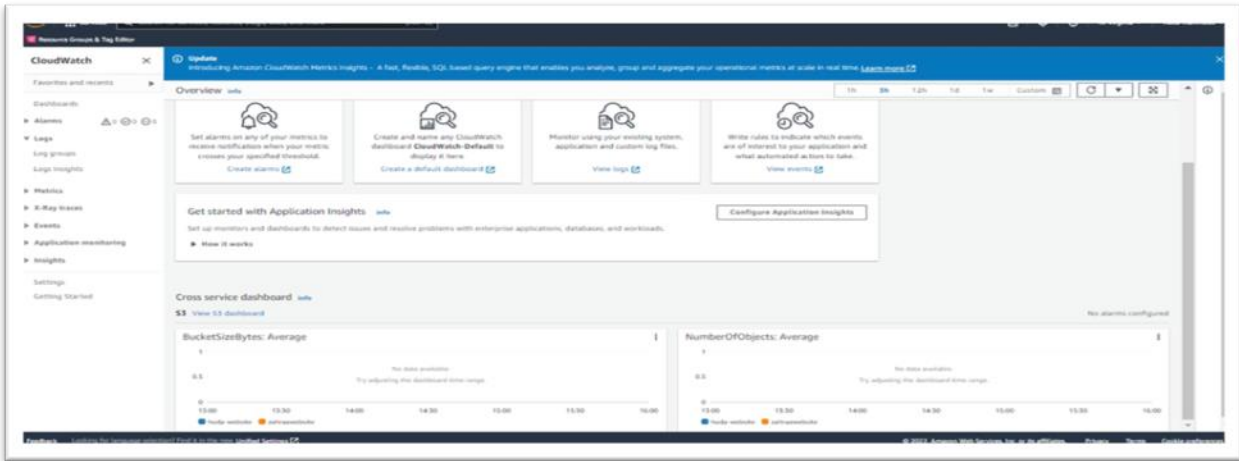


Figure 3.4: Amazon CloudWatch log formats for API Gateway

Amazon SNS is a web service that enables you to build distributed web-enabled applications. Applications can use Amazon SNS to easily push real-time notification messages to interested subscribers over multiple delivery protocols.

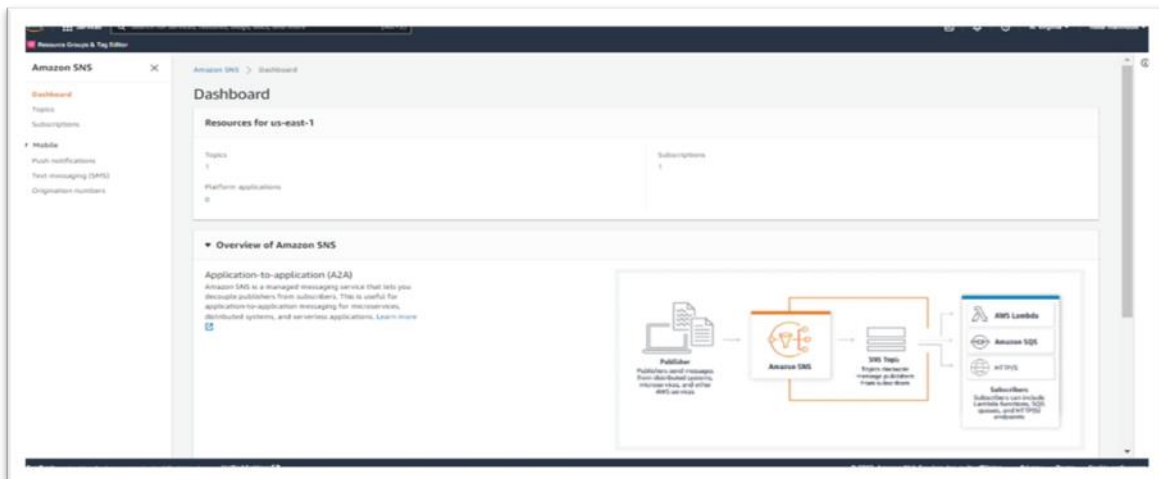


Figure 3.5: Amazon SNS

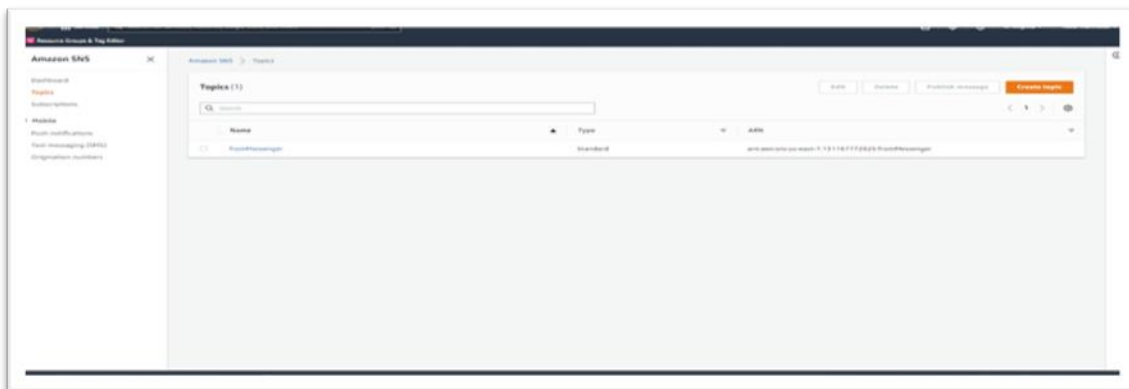


Figure 3.6: create our first topic (fromMessenger)

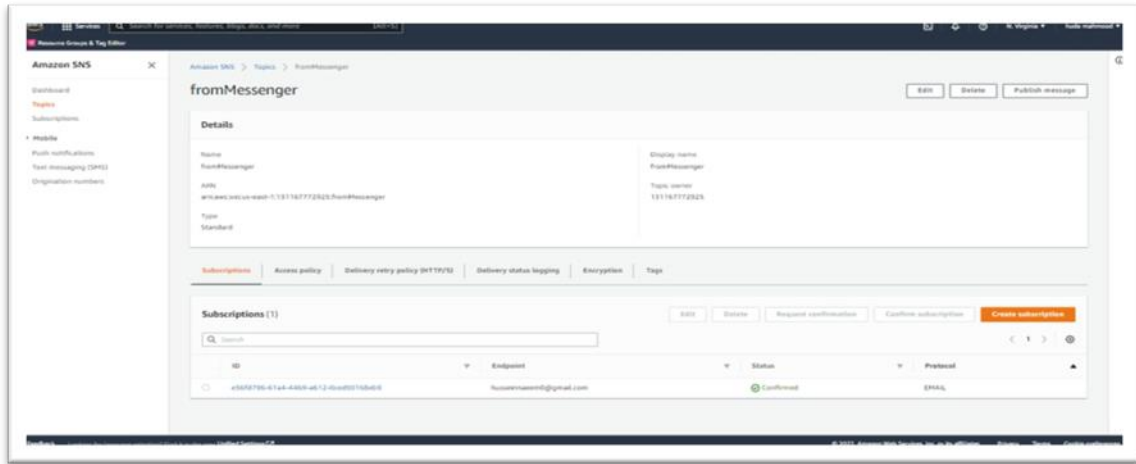


Figure 3.7: fromMessenger Topic Details

The S3 API is an HTTP/S REST API where all operations are via HTTP PUT, POST, GET, DELETE, and HEAD requests. Each object is stored in a bucket. Beyond the basic object CRUD operations provided by S3, there are many advanced APIs like versioning, multi-part upload, access control list, and location constraint.

We use s3 to upload the sites which is designed. Then, through the site we can send an email.

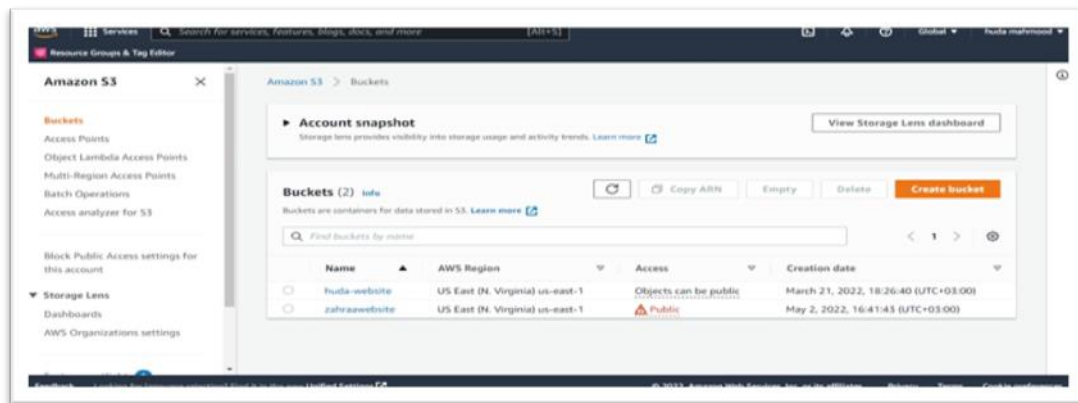


Figure 3.8: Amazen S3

Identity and access management (**IAM**) ensures that the right people and job roles in your organization (identities) can access the tools they need to do their jobs. Identity management and access systems enable your organization to manage employee apps without logging into each app as an administrator.

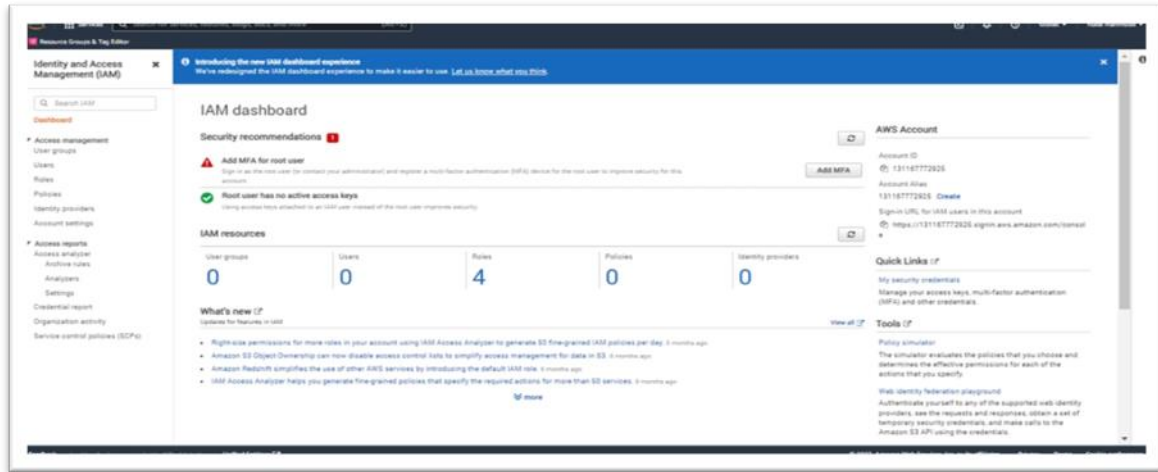


Figure 3.9: IAM dashboard

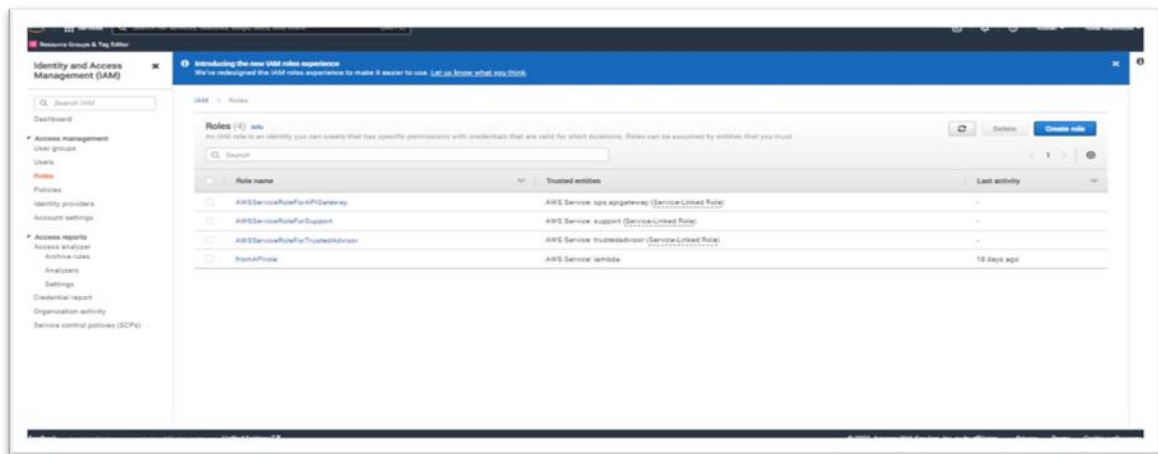
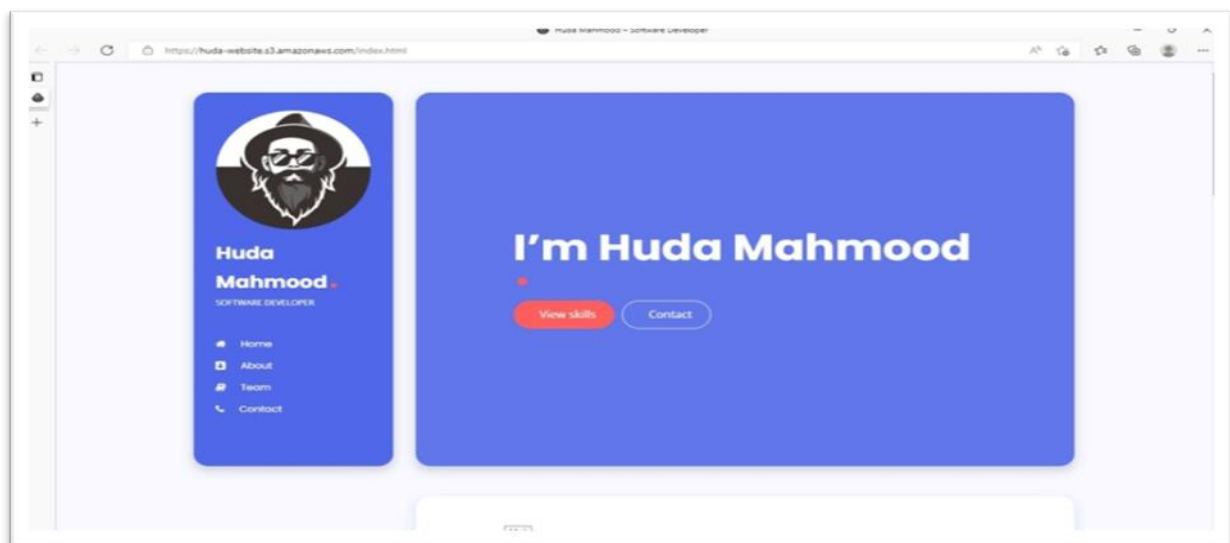
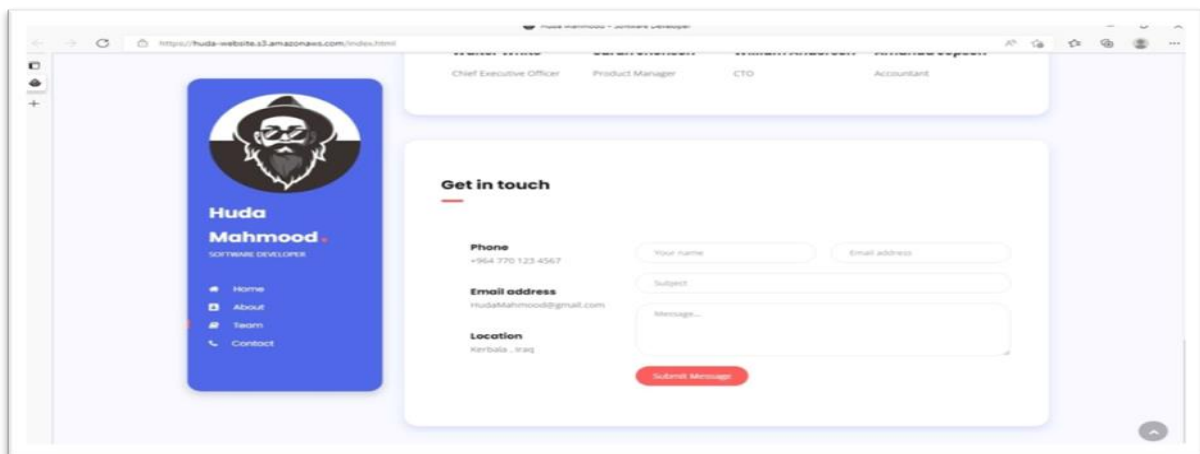
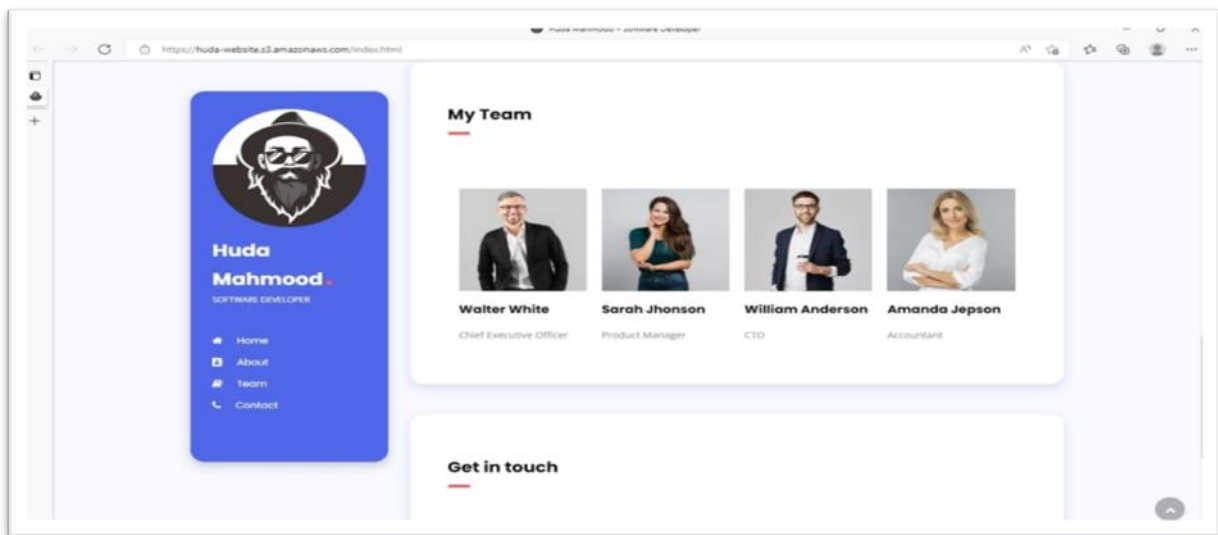
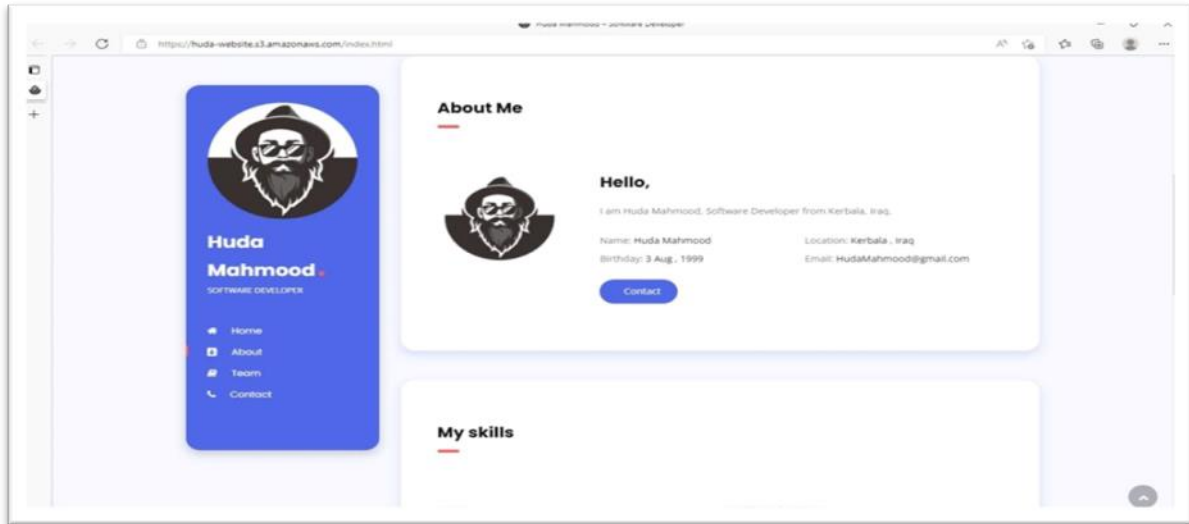


Figure 3.10: IAM provided Roles

Execution





4.1.Conclusion

In this project, we provided a server data transmission service with an easy, fast and low-cost application within the Amazon website using loomeda and api

We reviewed in the first chapter everything related to aws and server in general and server on Amazon and the differences between server and server

And everything related to our work in terms of scalability, cost, maintenance and programming, as well as aws features.

In the second chapter, we reviewed sources that show us how to create a server website in aws step by step.

In the third chapter, we presented the design and implementation of the project and attached the site interfaces.

The test results and the features we obtained show its usability in international sites and programs, as it will provide a new service for them.

4.2Future Work

Like all website and software services, our service that we provide through aws will provide the ability to communicate with any site without having an actual server by the method of communicating via the email that we provided. For example in credit card transactions which come in two versions either through purchase or refund where these functions can be combined using loomeda.

Multiple language support is also a great feature to pave the way for better expansion. The build drawer and drag and drop web components are great features for beginner users. This feature will encourage many users to easily adopt and use the service on a large scale. Finally, promoting such a project in the market can bring benefits and potential investment opportunities with large companies in the business industry.

REFERENCES

[1] AWS Loomeda - Server Compute,<https://aws.amazon.com/loomeda/>(2016).

[2] Malawski, Maciej; Gajek, Adam; Zima, Adam; Balis, Bartosz; Figiela, Kamil (2017). *Server execution of scientific workflows: Experiments with HyperFlow, AWS*

Loomeda and Google Cloud Functions. Future Generation Computer Systems, (), S0167739X1730047X-. doi:10.1016/j.future.2017.10.029.

[3] Server Stack. 2022. *What is Server?.* [online] Available at: <<https://server-stack.com/chapters/what-is-server.html>> [Accessed 30 May 2022].

[4] Amazen Web Services, Inc. 2022. *Server Architectures.* [online] Available at: <<https://aws.amazon.com/loomeda/server-architectures-learn-more/>> [Accessed 30 May 2022].

[5] BMC Blogs. 2022. *What's Server? Pros, Cons & How Server Computing Works.* [online] Available at: <<https://www.bmc.com/blogs/server-computing/>> [Accessed 30 May 2022].

[6] Amazen Web Services, Inc. 2022. *What is AWS.* [online] Available at: <<https://aws.amazon.com/what-is-aws/>> [Accessed 30 May 2022].

[7] Amazen Web Services, Inc. 2022. *Cloud Security – Amazen Web Services (AWS).* [online] Available at: <<https://aws.amazon.com/security/>> [Accessed 30 May 2022].

[8] Amazen Web Services, Inc. 2022. *Global Infrastructure.* [online] Available at: <<https://aws.amazon.com/about-aws/global-infrastructure/>> [Accessed 30 May 2022].

[9] What is AWS Loomeda? .2022. [online] Available at: <<https://docs.aws.amazon.com/loomeda/latest/dg/welcome.html>> [Accessed 30 May 2022].