DALHOUSIE UNIVERSITY

Title: Final Report for Term Project

Class: CSCI 5409 Adv. Cloud Computing

Professor: Robert Hawkey

Student: Jaivik Chetankumar Tailor

Banner ID: B00915987

Date: 12th April 2023

- ## Application Overview:

A to-do list web application is a software tool designed to help users keep track of their daily tasks and responsibilities. It allows users to create and organize their to-do lists, set due dates and reminders, and track progress toward completion. As a student, I must manage all my study tasks such as tutorials, assignments, and quizzes according to their deadlines. So, this kind of To-Do-List maker helps students to manage their studies effectively. Not only for students, but To-Do-List applications are also popular among individuals and teams who need to manage multiple tasks and deadlines in their personal and professional lives. They can be used for a wide range of purposes, from managing daily errands and household chores to planning complex projects and coordinating team activities. The application typically includes a user-friendly interface that enables users to quickly add new tasks and manage their existing ones.

- ## Features:

1. Sign up:

Users have two options during this process: either they can register for an account or just log in. In order to create a new account, the customer must enter basic information such as name, email, contact information, etc. and choose a new password. They can access their account after successfully creating one.

2. Log in:

The users can access the portal after setting up an account. The consumer has the option of adding new tasks or checking all the ones they have already added. Users have the choice of adding their work manually or via photograph.

3. Add To do Lists:

The to-do list web application provides users with a convenient and efficient way to add tasks, either by manually entering details or by uploading a photo. The user interface is designed to be user-friendly and intuitive, allowing users to add tasks and associated details quickly and easily.

To manually add a task, users simply need to enter the task name, description, reminder date, and due date in the corresponding fields.
Once the details are entered, the task is added to the user's to-do list, where they can easily view, edit, or delete it as needed.

For users who prefer to add tasks by photo, the application provides a convenient feature that allows them to take a photo of a task list or written note and upload it to the application. The application will then automatically extract the text and create new tasks based on the content of the photo and they can edit or submit their tasks. This feature is particularly useful for users who prefer to write down their tasks on paper but want to keep a digital record.

4. View To do Lists:

The "View All List" feature is a key component of the to-do list web application, which allows users to easily access and manage all their tasks in one place. This feature provides a comprehensive view of all tasks, so that users can quickly see what tasks are coming up and prioritize their workload accordingly. Additionally, users can edit or delete tasks directly from this view, without the need to navigate to individual task pages. This feature is particularly useful for users who have a large number of tasks or multiple ongoing projects, as it provides a bird's-eye view of all tasks in one place.

5. Edit/Update To do Lists:

The ability to edit and update tasks is a key feature of the to-do list web application, which allows users to make changes to their tasks as needed. To edit a task, users can simply click on the task name or details to open the task page, where they can make changes as needed. Users can modify any aspect of the task, including the task name, description, due date. Once the changes are made, the task is automatically updated and reflected in the user's to-do list.

6. Delete To do Lists:

The ability to delete tasks is an essential feature of the to-do list web application, allowing users to remove completed or unnecessary tasks from their list. This feature helps users stay organized and focused on their current workload by removing distractions or outdated tasks.

7. Logout:

This feature is a crucial aspect of the to-do list web application, allowing users to securely sign out of their account and protect their personal information. Logging out of the application ensures that unauthorized users cannot access the user's account and data.

- **Analysis:**

1. Services I used in my application and How I met item requirements?

- **List of services I used:**

  Compute: AWS Elastic Beanstalk and AWS Lambda
  Storage: AWS DynamoDB
  Network: AWS Virtual Private Cloud (VPC)
  General: AWS SNS and Amazon Textract

- **Analysis and Reasons:**

  First, **Elastic Beanstalk (EB)** streamlines the deployment procedure by automatically managing the deployment of the application stack, including load balancing, auto scaling, and other infrastructure parts. This frees developers (I) from worrying about setting up and managing the underlying infrastructure so they can concentrate on developing code and creating features for their application. EB provides a high level of scalability and availability by automatically scaling the application based on the current workload and providing fault tolerance and self-healing capabilities. This can help ensure that my application remains highly available and performs well even during periods of high traffic or unexpected spikes in workload. Setting up and administering an **EC2 instance** can be difficult, especially for newcomers like me to the cloud. It requires experience in infrastructure management, such as creating virtual machines, implementing load balancing, and monitoring performance. This can be time-consuming, and complex compared to EB. Integration with other services in AWS is easy with EB over other compute services that mentioned in list.

  AWS Lambda is an ideal solution when we need to run a small piece of code or function in response to an event trigger without managing the underlying infrastructure. The same concept I have used in my application to send Email using AWS SNS that's why I have used Lambda Function. AWS Lambda is highly scalable, which allows us to handle a large number of email requests without managing the underlying infrastructure. This is essential in a To-Do List application to send email to users to set their next To-Do-List. A reason to not use Docker in my application is that make simple applications complex while we deploy that.

  I have used **DynamoDB** for data management, DynamoDB's flexible data model allows for storing structured, semi-structured, and unstructured data, which makes it easy to store and retrieve data for my application. DynamoDB provides low-latency access to data, which means that users can retrieve their To-Do-List items quickly and efficiently. DynamoDB is a highly secure service that provides

several security features, including encryption at rest and in transit, access control, and auditing. While **S3 and Aurora** are both excellent storage systems, they might not be the greatest option for my application. S3 is primarily intended for storing and retrieving huge files such as photos, movies, and documents, and may lack the capabilities required for storing structured data. In contrast, Aurora is a relational database service that may be more difficult to set up and manage than DynamoDB. Overall, AWS DynamoDB is a better fit than S3 or Aurora for this application.

/TODO : Add details about VPC/

I have added the functionality of "add task list by photos", Amazon Textract assists me to extract text from images, which populate the task name, description fields, etc. automatically because Amazon Textract is a machine learning service provided by AWS that can extract text and data from scanned documents, forms, and tables. I didn't find any alternative for this Amazon Textract in the given list. In my application, I want to send an email notification to all users at a particular time when users get an email to set their To-Do-List so they can work efficiently on their tasks, for this functionality, I found **AWS SNS (Simple Notification Service)** the best one. AWS SNS provides a flexible messaging system that can send messages to multiple subscribers in a variety of formats, including SMS, email, and push notifications. This allows me to send emails using my application for task reminders and notifications in the format that is most convenient for them.

2. Details of Deployment model:

I have used the public cloud deployment model because it enables simple and extensive access to the programmer because can be accessed from any location with an internet connection. This is especially useful for users who are traveling or working remotely because they can still access and update their to-do lists without being tied to a particular device or network. Additionally, a public deployment model can be more cost-effective than hosting the application on a private server, as the provider of the public deployment may offer a range of pricing plans to suit different needs and usage levels. This can be particularly advantageous for small businesses or individual users like students, who may not have the resources to maintain their own servers or pay expensive hosting fees. According to my application needs, the community cloud deployment model and private is not suitable and required.

3. Details of Delivery model:

To-Do List application is a Software as a Service (SaaS) application. According to the SaaS model , it is a cloud delivery model in which software applications are hosted and delivered over the internet by a third-party provider. Users access the application

through a web browser or API. In the case of my To-Do List application, it is delivered over the internet and hosted on a server provided by a third-party provider(AWS beanstalk and Heroku). Users can access the application through a web browser or API. Therefore, I mentioned that my application is a SaaS application.

I have used the Software as a Service (SaaS) delivery model for my To-Do List application because it provides several benefits that are well-suited for the needs of my users. By delivering the application over the internet, I can offer easy accessibility to my users who can access it from anywhere with an internet connection. Additionally, with SaaS, I can easily scale up or down my application to meet the growing needs of my users. Updates and maintenance are handled by the SaaS provider, freeing up my time and resources for other important tasks. Finally, SaaS providers typically implement robust security measures, which can be particularly beneficial for applications that handle sensitive or confidential information. Overall, using the SaaS model allows me to offer a better user experience, increase security, and scale my application quickly and efficiently, making it ideal for my users.

4. Final Architecture Description:

1. How do all the cloud mechanisms fit together to deliver your application?

The primary requirements of the web application are caused by the numerous cloud techniques. The cloud services used in the project's implementation are serverless computing, storage, network, and compute. All these services work together seamlessly to produce a complete cloud-based web application. The **AWS EB** service handles the compute part of the project for deployment to be made available remotely whereas **AWS DynamoDB** is used for the storage of images and all other information. Additionally, **AWS SNS** is used to remind users to make their lists, so these SNS services are used for sending notifications to users about their to-do-list. To send email, application first Tigger **AWS Lambda** function. **Amazon Textract** boosts the use of the application by adding a contemporary feature such as a to-do list from photos. That enhances the functionality of an application. Going on to the application's front end, it was created with Material UI and React.js. The programmer will be viewable by the user as a web-based service. In other words, the Amazon services will manage and maintain all of the **TO-DO-LIST-MAKER** application's essential features. The seamless integration that is offered between various services, creating a complete package, makes it feasible for all of these AWS services to work together in harmony. The **TO-DO-LIST-MAKER** frontend application allows users to add, view, and amend their tasks. The app provides user registration, user login, add To-Do-List, update To-Do-List, delete To-Do-List, add To-Do-List using photos, logout. The application's core functionalities, as listed above,

will be handled by the backend services. The APIs are used to establish the communication channel between the frontend and the backend. The data saved in the cloud is updated, fetched, and posted using these APIs. The data will be collected from the backend and rendered on the user interface for the user to view the requested data. The data will be posted or updated to the backend in DynamoDB with the aid of the UI. The integration of all cloud-based services with the responsive user interface establishes the process and data flow between the various application components.

2. **Where is the data stored?**

The Data in the To-Do-List-Maker website is tailored to store the user details. like the username, email address, password, and contact information. All these details get stored in the **AWS DynamoDB** with appropriate hashing for password. I selected AWS DynamoDB because it stores data across numerous servers using a distributed design, making it extremely available and durable. DynamoDB is intended to be a scalable NoSQL database capable of handling huge workloads, and it automatically splits data among servers to ensure constant, low-latency performance. Amazon also offers backup and restoration options for DynamoDB to ensure that data is safe and recoverable in the event of an outage. Any other data generated by the website is kept in **Amazon DynamoDB**, such as the user's To-Do-List information, including task name, task description, and task deadlines, for further analysis or visualization. **Amazon SNS** subscriber list is also triggered whenever a new user registers in the system. Similarly, **Amazon Lambda** was triggered at a predetermined time to get a user from a database associated with an AWS SNS subscriber list. Therefore, in essence, **AWS DynamoDB** is utilized to store the majority of the data used by the To-Do-List-Maker application.

3. **What programming languages did you use (and why) and what parts of your application required code?**

The front-end of the application is created with the React.js framework, while the backend is built with the Node.js and Express.js frameworks. As a result, JavaScript is the primary programming language utilized to create the To-Do-List-Maker application. Using JavaScript and framework libraries, the AWS cloud services were integrated with the application's frontend. There was also coding for the Amazon Lambda function, which is triggered programmatically at a set time. As a result, these were the components of the application that required programming.

There are several reasons to choose these two for my application. Together, ReactJS and Express Node.js provide a full-stack web development solution that enables developers to build scalable and performant web applications. ReactJS

handles the client-side rendering and user interactions, while Express Node.js handles the server-side logic and data management. This separation of concerns allows for a more modular and maintainable codebase. Additionally, both ReactJS and Express Node.js have large and active communities, which means there are plenty of resources and third-party libraries available to help you build your application efficiently.

4. **How is your system deployed to the cloud?**

In your project, I have used a combination of deployment models - **Heroku for the frontend** and **AWS Elastic Beanstalk** for the **backend.** This hybrid or multi cloud approach allows more efficient and reliable deployment of my application. I have used Heroku because is a cloud-based platform as a service (PaaS) that provides an easy-to-use interface for deploying, managing, and scaling web applications. It is particularly useful for the front end of an application as it provides a simple way to deploy web apps using preconfigured environments and dependencies.

On the other hand, Elastic Beanstalk is an AWS service that provides a scalable and flexible platform for deploying and managing our application's backend. I have deployed my application's backend on Elastic Beanstalk, which automatically handles the underlying infrastructure such as load balancing, auto-scaling, and monitoring. This allows us to focus on developing our application's logic without worrying about the underlying infrastructure.

Together, Heroku and AWS Elastic Beanstalk offer a powerful combination of ease-of-use and flexibility. Whether application needs a simple, straightforward deployment process or a more complex, scalable infrastructure, Heroku and AWS Elastic Beanstalk have the tools and capabilities that my application needs to deliver a high-quality application. A benefit of multi-cloud architecture is the ability to take advantage of the unique features and capabilities of each cloud provider. For example, Heroku offers better support for certain frontend programming languages or frameworks, while AWS Elastic Beanstalk offers better scalability and performance with backend languages like Java, Python, Nodejs, etc.

5. Final Architecture :

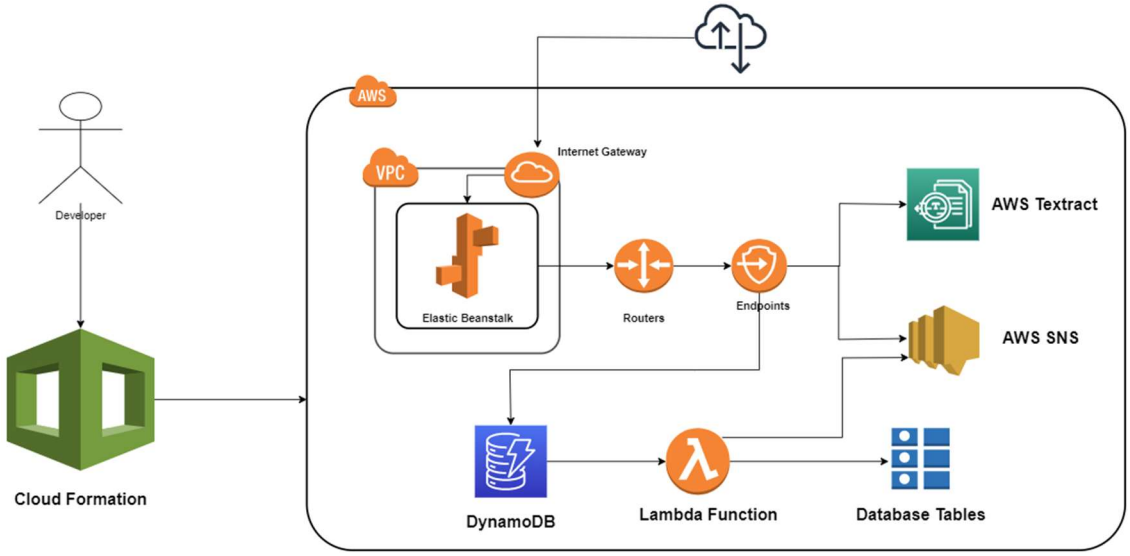

Figure 1 Application Architecture [1]

The above image illustrates the overall architectural flow of my application along with all the AWS services I have mentioned above.

Firstly, I created a Virtual Private Cloud (VPC) in AWS, which provides an isolated network environment for your resources to run in. Elastic Beanstalk environment, DynamoDB table, SNS topic, Amazon Textract, and AWS Lambda all are connected with this VPC. A VPC allows me to isolate my resources from the public internet, making them more secure. I have created security groups and route tables to control inbound and outbound traffic to and from my resources. By creating a VPC, I have complete control over my application's virtual networking environment. I can choose the IP address range for my application and VPC and create subnets within it. Not only that but I have also configured routing tables to control traffic between your subnets. API endpoints within my VPC for each of the services , I ensure that my resources are securely integrated and can communicate with each other without going through the public internet.

For deployment, I have used Elastic Beanstalk, which is a managed service that allows my application to easily deploy and scale web applications. Elastic Beanstalk automatically provisions the necessary resources to run my application, including EC2 instances, load balancers, and auto-scaling groups. In addition to that, I have used DynamoDB as your storage solution that provides fast and predictable performance at any scale for application. I have connected AWS SNS with AWS Lambda when the backend will trigger the lambda function then lambda will access the DynamoDB to get the user's data and send an email using the AWS SNS.

Overall, this architecture provides a secure and scalable environment for my web application to run in, with fully managed services for storage, messaging, and machine learning. The use of AWS Lambda adds serverless compute capabilities to my application, allowing me to execute code in response to events without managing servers.

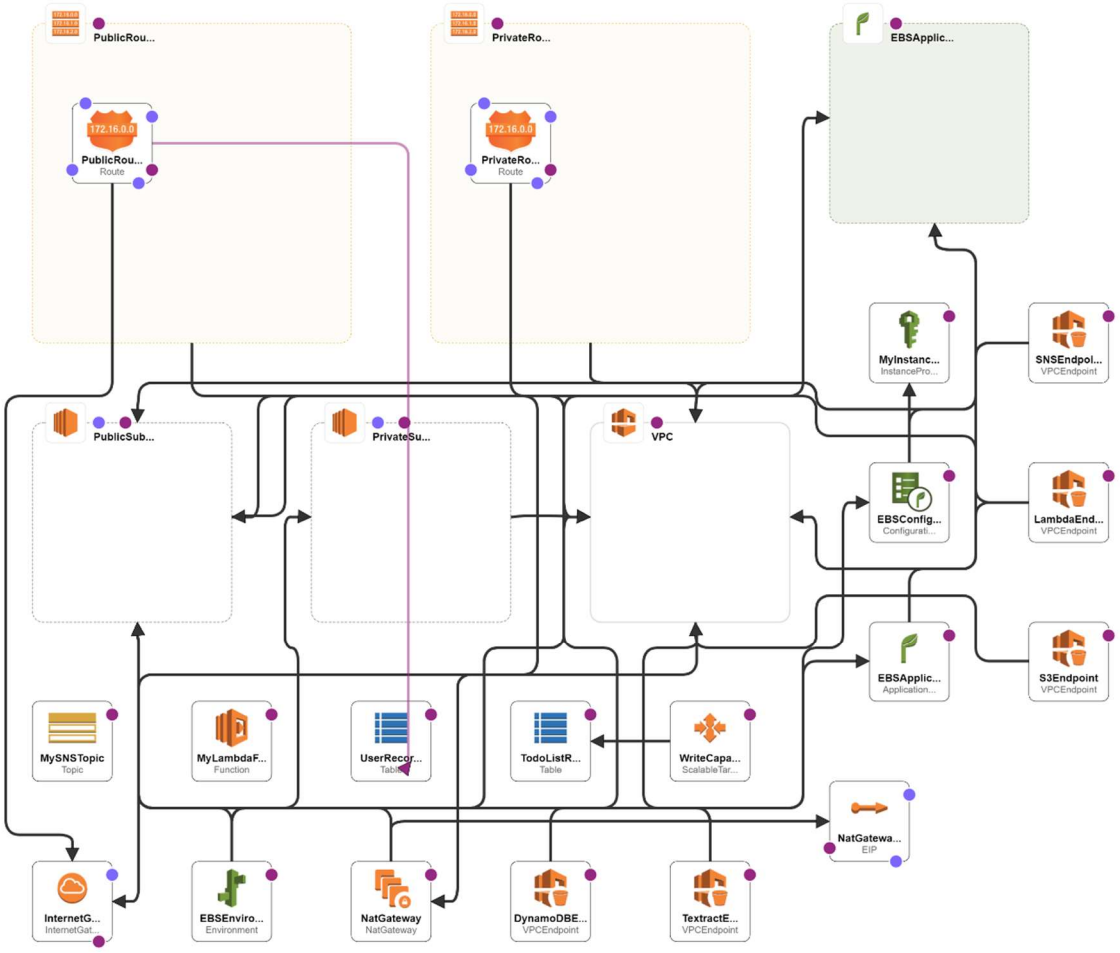

Figure 2 Architecture flow Created by Cloud Formation [2]

I have created this whole architecture using Cloud Formation only. Where I have used .yml file to make my whole application.

6. Data Security at all layers:

I have tried to achieve data security at each layer in a different way. For Network Layer, I've deployed my application, database, and other resources within a **Virtual Private Cloud (VPC),** which provides a private, isolated network environment within AWS. I have also used security groups and network access control lists (ACLs) to control traffic to and from my resources, and I have also configured routing tables to control traffic between the system's subnets. I used secure coding practices for the application layer, such as input validation, output encoding, and password hashing. I have also utilized HTTPS to secure traffic between your application and clients. Regarding storage layer, I stored user's data in DynamoDB. I have also utilized DynamoDB encryption client to ensure that data is encrypted before it is written to the database. Additionally, I have implemented access controls and permissions at the database level to ensure that only authorized users have access to sensitive data. I deployed and managed my application using Elastic Beanstalk at the compute layer, which automatically handles compute resources including EC2 instances and load balancers. In a serverless environment, I have also used Amazon Lambda to run code, which offers automatic scaling and high availability without requiring further management of the underlying infrastructure. This makes application more secure. SNS and Textract are two examples of AWS services I may use to connect my application to other services. I have set up API endpoints inside of VPC for each service, preventing arbitrary users from directly accessing services over the internet. There are no such vulnerabilities for first level systems because I have included network isolation, secure coding principles, encryption, access limits, and safe computation and integration.

7. Cost Based Aspects:

While navigating the application, I came across a variety of cloud services that have a cost associated with them. However, the severity of cost-cutting from our AWS academy account varies depending on the services we use. The services we use produce an estimate cost of as listed below.

- DynamoDB: DynamoDB Streams cost $0.02 per 100,000 read operations. Outside of the AWS region where the DynamoDB database is installed, data requests are charged $0.09 per GB . That should be roughly $0.0002 based on our usage of building projects and implementing queries.

- AWS SNS:
    - Unit conversations:

- HTTP/HTTPS Notifications: 1 million per month * 1000000 multiplier = 1000000 per month [3]
- Pricing calculation:
  - Max (0 requests, 0 requests) = 0.00 requests
  - Tiered price for: 1000000 calls [3]
  - 100000 calls x 0.0000000000 USD = 0.00 USD (first 1 million free) [3]
  - 900000 calls x 0.0000006000 USD = 0.54 USD
  - Total tier cost: 0.00 USD + 0.54 USD = 0.54 USD (HTTP/HTTPS Notifications cost)

Right now, in the building mode our application does not exceed 1 million counts. So, it was a relief as per the cost from SNS.

- AWS VPC: Among all the services we used, I believe VPC was the most expensive. For an hour, it costs $0.045 per Nat gateway. The amount we used from our academy account is difficult to estimate for VPC, but it may be between $3 and $10. [3]

- AWS Lambda: The free tier, like the SNS, gives us 1 million free first requests, which was sufficient for constructing and testing. Nonetheless, there is a possibility of removing only a few credits from our accounts.

- Amazon Textract: This service is free for up to 100 billing pages. After that 100 billable pages with text x 0.0015000000 USD = 0.15 USD [3]. So, for the initial case it can be free but after some time , it will create cost issues.

- AWS Elastic Beanstalk: Elastic Beanstalk running an EC2 instance on t2.micro in the US East (N. Virginia) region for my application.

  - EC2 t2.micro instance cost: $0.0116 per hour (based on on-demand pricing as of April 2023) [3]
  - Elastic Beanstalk platform fee: $0.025 per hour for Linux/UNIX instances [3]
  - Data transfer in: Free [3]
  - Data transfer out: $0.09 per GB [3]
  - Storage: $0.05 per GB-month for Amazon EBS General Purpose SSD storage [3]

  If our application runs continuously for a month with no additional services used, the estimated cost would be:

  - EC2 t2.micro instance cost: $8.38/month

- Elastic Beanstalk platform fee: $18/month
- Storage (assuming 1 GB usage): $0.05/month
- Total: $26.43/month

We can better grasp more affordable alternatives to what we have used thus far after reflecting on the compatibility of the services we are utilizing with their consumption costs. We have decided on the services for our AWS academy accounts that wouldn't get pricey. However, there are methods by which we can continue to cut expenditure. We may have used areas with affordable service costs. A small amount of manipulation by changing regions can be a helpful way to reduce costs as the programmer is still only being created as a prototype. When compared to selecting regular size services, using small size instances, buckets, and services also results in cost savings. Furthermore, we focused on lambda usage, which should exceed the level where charges can become prohibitively expensive. Removing the resources when we had completed our tasks helped us save money as well. Delete unattached elastic IP addresses, delete instances before closing down the lab, and so on.

8. Future Aspects:

Currently in my application users can make their To-Do-List using a manual approach and by photo, delete their task and update that accordingly. If we look at the future of development, I would like to implement the following things in my application that would use the following cloud mechanisms as mentioned below.

- **Chatbot**: I would use **Amazon Lex** to create a chatbot interface for my To-Do-List maker. This would allow users to interact with the application using natural language commands.

- **Sorted To-Do List Notifications**: I would use **AWS SES**, that will send personalized mail to user with their already added tasks, so they never miss their deadlines and be vigilant.

- **Customized Email Reminders**: I would like to make customized email reminders for users when they want reminders for their task, for example, before 2 days of their task deadline. I will use a combination of **AWS Lambda and AWS SES** to develop this.

- **Tasks Visualization**: You will use **Amazon QuickSight** to create dynamic and interactive visualizations of your user's to-do lists. This could help users

better understand their progress towards their goals, identify patterns in their behavior, and make data-driven decisions about their tasks.

References:

[1] "Draw.io," JGraph Ltd, [Online]. Available: https://app.diagrams.net/. [Accessed 10 April 2023].

[2] "AWS Console," Amazon, [Online]. Available: https://us-east-1.console.aws.amazon.com/console/home?region=us-east-1. [Accessed 11 April 2023].

[3] "AWS Pricing calculator," Amazon, [Online]. Available: https://calculator.aws/#/. [Accessed 11 April 2023].