Master of Applied Computer Science

(CSCI 5409: Advanced Cloud Computing – Summer2023)

Term Assignment

Instructor
Robert Hawkey

Banner ID: B00919946 | **Name:** Sarthak Patel | **Email ID:** **sr555161@dal.ca**

Sarthak Patel (B00919946)

Expense Analyzer Application

I. What I Built and Its Purpose

I have developed an Expense Analyzer Application, a comprehensive cloud-based expense tracking and analysis platform. The application's primary purpose is to provide users with a streamlined way to manage their expenses efficiently. As of now, the user of the application is only me. So basically, it is a personal expense analyzer application. The main features of the application include:

- **Receipt Upload and Analysis:** Users can upload images of their receipts to the application, which are then processed using AWS Textract. Textract extracts relevant information from the receipts, such as invoice date, vendor name, total amount, etc.

- **Expense Storage:** The receipts uploaded by the users are stored in S3 and are provided with download link along with the expense in the frontend, so that they can see their receipts anytime. The extracted information is stored securely in AWS DynamoDB, forming structured expense records for each receipt.

- **Expense Display and Analysis:** Users can access their expense data through an API Gateway integrated with AWS Lambda. The API Gateway triggers the relevant Lambda function to fetch the data from DynamoDB and display it in a user-friendly tabular format.

- **Email Notifications:** The application utilizes AWS SNS (Simple Notification Service) and AWS EventBridge to send weekly email notifications to users, summarizing their spending patterns.

II. Meeting Menu Item Requirements

To build this application, we used the following AWS services, along with a comparison of alternative services and the reasoning behind our choices:

II.I Compute

- **Which services used?**

  a. Lambda

  I chose AWS Lambda for the backend because it offers a serverless architecture, which eliminates the need for managing servers. It makes it easy to process receipt images, store data in DynamoDB, and send email notifications. This enables cost-effectiveness, auto scaling, and easy deployment of functions, making it an ideal choice for my expense analyzer application. Also, Easy to integrate with other services with the help of triggers and events.

  b. EC2

  I opted for AWS EC2 because it provides full control over the virtual machines, allowing me to customize the environment to suit your frontend requirements.

Additionally, EC2 offers scalability and reliable performance, making it suitable for hosting the frontend of your application.

- **Why not others?**
  - **AWS Elastic Beanstalk and Docker:** No specific reason for not choosing, I chose EC2 over Elastic Beanstalk because I was familiar with the EC2 due to previous assignments, and I just had to replicate the steps which I did in assignments.

  - **AWS Elastic Container Service (ECS) and Amazon EKS:** Access not given in AWS Academy.

  - **AWS Step Functions:** Overkill for the simple backend operations of the expense analyzer application.

  - **AWS IoT 1-Click:** Not applicable for the functionality required in the expense analyzer application.

II.II Storage

- **Which services used?**

  a. S3
  S3 was chosen for receipt storage due to its simplicity, durability, and easy integration with other AWS services. It provides a highly durable and available storage solution for any type of data, including images, documents, or videos (not using in my application).

  b. DynamoDB
  Frankly, I am little bit biased towards NoSQL database because it provides the feature to store unstructured data. Secondly, I opted for DynamoDB as the NoSQL database to store expense records. Its fast and scalable nature suited our application's dynamic schema requirements, and its seamless integration with Lambda made it a favorable choice.

- **Why not others?**
  - **AWS Aurora:** Aurora, being a managed relational database, might be overkill for the application's data requirements, and the costing is much high compared to using DynamoDB [1].

  - **AWS Athena:** I am not a fan of SQL Databases as they make things unnecessary complex, also, Athena is primarily used for SQL querying data stored in S3, which is unnecessary for the application's data access patterns [2].

- **AppSync:** AppSync's GraphQL autoscaling and data synchronization capabilities are not essential for the straightforward data storage and retrieval needs of the expense analyzer application [3].

- **IOT Analytics:** IoT Analytics is designed for handling data from IoT devices, which is not relevant to the expense analyzer application.

- **Neptune:** Neptune is optimized for handling graph data, so beyond scope of project [4].

- **Relational Database Service:** RDS is a managed relational database, but DynamoDB's NoSQL capabilities align better with the application's data model and requirements. Additionally, DynamoDB's seamless scalability is beneficial for handling potential increases in data volume.

II.III Network

- **Which services used?**
  a. **API Gateway**
  API Gateway provides a secure and scalable way to manage and route API requests to AWS Lambda functions or other backend services, making it an excellent choice for handling frontend-backend communication in the expense analyzer application. Also, easy to put trigger with lambda.

- **Why not others?**
  - **VPC:** No specific reason, I could have used VPC to restrict and secure my DynamoDB database, but I eventually would have to use API Gateway for Backend and Frontend communication. So, I just skipped to use VPC. I might use in future.

  - **CloudFront:** CloudFront is a content delivery network (CDN) used for high-speed content delivery and improving the performance of web applications by caching and distributing content to edge locations [5]. While CloudFront can enhance user experience, it's not a primary requirement for my expense analyzer application**.**

II.IV General

- **Which services used?**
  a. **AWS Textract**
  I used AWS Textract because it provides powerful text recognition capabilities, allowing me to extract text and data from scanned documents, images, and PDF files. With Textract, I can automatically process and analyze receipts uploaded by users, extracting relevant information such as vendor names, invoice dates, and total expenses. This helps automate the data extraction process, making it efficient and accurate, thereby enhancing the user experience and reducing manual data entry efforts.

b. **AWS SNS (Simple Notification Service)**
I used AWS SNS to send email notifications to users regarding their weekly spendings. By integrating SNS with Lambda functions, I can easily trigger email notifications every week with the help of EventBridge.

c. **AWS EventBridge**
I used AWS EventBridge to build an event-driven architecture for my application. With the help of EventBridge, I can send notifications to the user for their weekly expenses as well as an alert message if they have set any threshold amount in their expense (will implement in future.)

III. Deployment Model

The application is deployed in the cloud using Infrastructure as Code (IaC) with the help of CloudFormation and follows a multi-tier architecture. The components are hosted on AWS, ensuring high availability, scalability, and ease of management. So, the overall, deployment model of my application expense analyzer is **Public Cloud**.

IV. Delivery Model

In my project, I utilized a combination of AWS's Infrastructure as a **Service (IaaS) and Function as a Service (FaaS) services**. The front end of my application was deployed using EC2, an IaaS service. This allowed me to have complete control over the infrastructure, including the operating system and runtime environment, providing more configuration and administration options.

For the backend logic, I chose Lambda, a FaaS service. With Lambda, I could focus solely on writing my code while AWS managed the underlying infrastructure, such as the operating system and runtime environment. This resulted in lower operational expenses and faster development and deployment timelines.

The flexibility of EC2 allowed me to modify the infrastructure to meet specific requirements. For example, I initially selected the t2. micro instance with 1 GB of RAM for the frontend server, but it crashed due to a memory out-of-bounds error. I then switched to the t2. small model with 2 GB of RAM, resolving the issue. However, it was still slow, so then I switched to t2. Medium as a result it increased the performance of the application. Meanwhile, using Lambda freed me from infrastructure management worries.

The combination of IaaS and FaaS services provided my project with a robust and adaptable deployment approach. Well, right now it is not a Software as a Service (SaaS) but by adding signup login functionality and doing business to business license agreement with the users it can be used as a Software as a Service in the future..

V. System Architecture

The system architecture consists of the following components:

V.I. Architecture Flow

The application uses the IP address of the EC2 instance provisioned from the CloudFormation along with the port 3000 of the react to run the expense analyzer application. The user will upload the bill or receipts either in image or pdf format. The object uploaded will be stored in S3 Bucket which will trigger the Receipt Extractor Lambda, responsible for extracting the information with the help of AWS Textract. The extracted information will be stored in DynamoDB. After the object has been successfully uploaded in S3 and information extracted, it will trigger API Gateway URL in frontend to fetch the expense uploaded along with recent expenses. The whole process can be seen in figure 1.

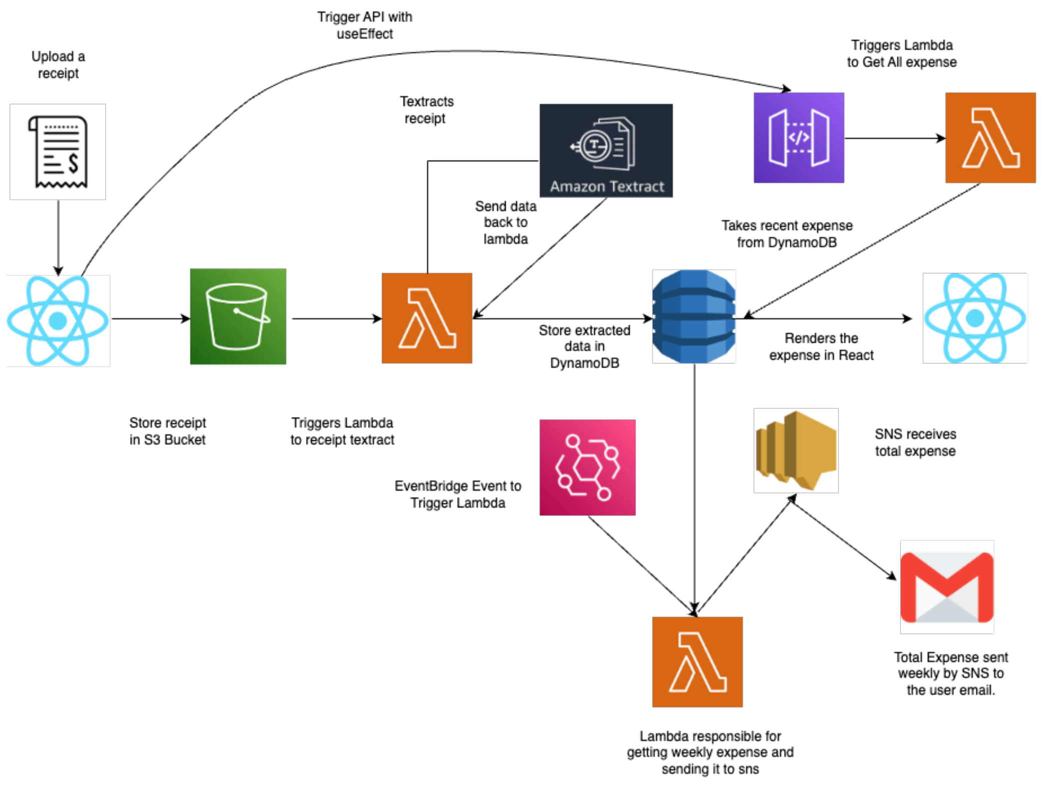
Trigger API with useEffect
Upload a receipt
Triggers Lambda to Get All expense
Textracts receipt
Amazon Textract
Send data back to lambda
Takes recent expense from DynamoDB
Store extracted data in DynamoDB
Renders the expense in React
Store receipt in S3 Bucket
Triggers Lambda to receipt textract
SNS receives total expense
EventBridge Event to Trigger Lambda
Total Expense sent weekly by SNS to the user email.
Lambda responsible for getting weekly expense and sending it to sns

Fig. 1: Architecture flow diagram of expense analyzer application [6]

V.II. Data Store

- For Data Layer or Data Storage, I have used S3 and DynamoDB to store the receipts and extracted information from the receipts respectively. To give an analogy, it can be assumed as storing unprocessed or raw data in S3 and processed data in DynamoDB, so both data's can be conserved for the users. That way user can see the receipt details and receipt both in frontend.

V.III. Programming Languages

- **JavaScript:** JavaScript has been used both in Frontend for ReactJS and in backend in lambda functions. Material UI has been used for styling and to make the website responsive. For the backend NodeJS has been used to work around as I am more comfortable with Node JS compared to python or other languages.

- **Shell Script:** Shell Script has been used in the cloudformation.yaml file. It is responsible for installing node JS as well as AWS CLI in the EC2 Instance once provisioned. AWS CLI has been installed to access the S3 Bucket Object where I have uploaded my Frontend code. And installing node JS to install the node dependencies and running the React Application in EC2 instance. Instead of AWS CLI, other option was to upload the code on GitHub, install git on EC2 instance and access the code from there but I would have to make my repository public which I didn't want to do. So, I went with this approach.

- **YML Script:** YAML script has been used for the CloudFormation to allocate AWS resources instead of JSON file because YAML file is more popular than JSON and has more documentation related to it.

- Every part of the application required code. Code for infrastructure or provisioning services as well as code for the frontend ReactJS.

V.IV. Cloud Deployment

Entire application is deployed to the cloud. I just have to upload my CloudFormation.yaml file in AWS CloudFormation Service and every AWS resource used in the project will be allocated on its own. I would just have to take the IP address of the provisioned EC2 and run it with the port 3000 to access my application. The communication between frontend and backend is done through API Gateways.

V.V. Comparison with the architecture taught in the course

- **Workload Distribution Architecture:** My architecture doesn't fall in this category because I haven't used load balancers to distribute my workload. I have used EC2, and it has static resource allocated.
- **Dynamic Scaling Architecture:** Same reason as Workload Distribution Architecture. In addition, I haven't used automated scaling listener.
- **Resource Pool Architecture:** I haven't used the concept of one or more resource pools and its synchronization.
- **Cloud Burst Architecture:** My application is fully on cloud. So out of scope.

- **Elastic Disk Provisioning Architecture:** In my application I didn't use Elastic Block Storage (EBS) for the elastic disk provisioning, instead I used DynamoDB to store the data because it is NOSQL as well as offers low latency, durability, and scalability.
- **Dynamic Data Normalization:** Redundant Data hasn't been managed.
- **Redundant Data Storage:** No concept of primary and secondary storage device. Only one data storage.
- **Cloud Balancing Architecture:** My application is Single cloud application, so no need of cloud balancing.
- **Load Balanced Virtual Server Architectures:** No concept of watchdogs to see my virtual server or EC2 instance.
- **Rapid Provisioning Architecture:** It can be considered as a Rapid Provisioning Architecture because by using infrastructure as a code, I was rapidly able to provision the resources in the AWS and could use my application.

VI. Security Approach

- The data has been tried to keep secure at almost layers.
- In Transit: HTTPS is used to encrypt data transmission between clients and the server, ensuring secure communication.
- At Rest: Sensitive data, such as receipts and expense records, are encrypted using AWS managed encryption services.
- In compute layer, data has been kept secure with the help of assigning a security group to the instance, by defining the inbound and outbound rules to the instance. As well, RSA key has been generated for the EC2 instance.
- In Network Layer, API Gateway has been used to secure the endpoints by using hypertext transfer protocol secure.
- In storage layer, DynamoDB has been kept secure with the help of IAM users.
- Data processed by Textract during receipt analysis remains confidential and is automatically deleted after processing from the AWS Textract.
- As of now, there is only one place the security can be compromised. I haven't created the signup login in my application, so if any person get application link, they can see my expenses or application, but it is for my personal use, So I would not share with anyone. However, I plan to continue this project by adding signup login soon.

VII. Cost Metrics Analysis

The cost of operating the system includes:

- **Virtualization Software:** The cost of virtualization software, like VMware, can vary depending on the number of licenses needed. Cost of one standard VMware is around $2,995 and if the company requires at least five licenses, the price could be $15,000 [7].
- **Networking Components:** To set up a network for the private cloud we need to set up the whole network and for that we need switches, routers, modems, load balancers Surge

Protector, ISP’s, Network Firewall, Battery Backup, Cables, and connectors and many more then it would cost around $5,537 in 2023 [8].

- **Database Setup:** We will need to set up our own NoSQL database infrastructure like DynamoDB if we want to replicate. Things need to be taken for the disaster recovery. So, one can expect around $2,000 to $10,000 for small business and larger companies respectively [9].
- **OCR Setup:** High quality OCR software licenses can range from $2,000 to $5,000. Hardware can be around between $5,000 to $10,000. So overall cost can be expected around $10,000 to $15,000 (based on assumption as couldn’t find resource).
- **Emails and Notifications:** The organization would need to purchase ad setup email server software, and this could range from $3,000 to $10,000. Investments in email security can range from $1,000 to $2,000 (based on assumption as couldn’t find resource).

VIII. Monitoring

The cloud mechanism that would be most important for me to add monitoring to make sure costs do not escalate out of budget unexpectedly would be EC2 compute service responsible for running the frontend of the application. The reason for this is that EC2 is an Infrastructure as a Service (IaaS) offering, which means it provides virtual machines that need to be running continuously to host the application's frontend. Unlike serverless services, such as AWS Lambda, which only incur costs based on actual usage, EC2 instances are billed per hour of usage, regardless of whether they are actively processing requests or idle.

IX. Application Evolvement

- **Authentication:** Signup and Login functionality can be added to make it multiuser.
- **Graphical Display of Expenses:** Other Service like AWS Quick Sight can be incorporated to display the expenses graphically.
- **Expense Categorization and Tags:** Introduce features to automatically categorize expenses based on keywords or allow users to tag expenses for easier tracking and analysis using AWS Comprehend.
- **Budgeting and Spending Insights:** Add budgeting functionality that enables users to set spending limits and receive alerts when nearing or exceeding them using SNS. Provide spending insights and suggestions for better financial planning using.
- **Multi-currency Support:** Allow users to record expenses in different currencies and provide real-time currency conversion for accurate tracking.
- **Machine Learning for Smart Analysis:** Utilize machine learning algorithms to analyze spending patterns, identify trends, and offer personalized money-saving tips to users.

X. Citations

[1] “DynamoDB vs Amazon Aurora - the ultimate comparison,” Dynobase, https://dynobase.dev/dynamodb-vs-aurora/ (accessed Aug. 1, 2023).

[2] “Amazon Athena vs amazon dynamodb: What are the differences?,” StackShare, https://stackshare.io/stackups/amazon-athena-vs-amazon-dynamodb (accessed Aug. 1, 2023).

[3] D. Dulal, “DynamoDB graphql API’s in AWS AppSync,” Medium, https://articles.wesionary.team/dynamodb-graphql-apis-in-aws-appsync-6663d0b24607 (accessed Aug. 1, 2023).

[4] “DynamoDB vs Amazon Neptune - the ultimate comparison [2023],” [2023], https://dynobase.dev/dynamodb-vs-amazon-neptune/ (accessed Aug. 1, 2023).

[5] A. Soni, “Understanding and using Amazon Cloudfront CDN,” Medium, https://medium.com/mindful-engineering/today-we-will-learn-about-cloudfront-690bf3a8819a (accessed Aug. 1, 2023).

[6] “Draw.io - free flowchart maker and diagrams online,” Flowchart Maker & Online Diagram Software, https://app.diagrams.net/ (accessed Aug. 1, 2023).

[7] J. Brodkin, “VMware’s hypervisor is free, but enterprises will still pay,” Network World, https://www.networkworld.com/article/2274360/vmware-s-hypervisor-is-free--but-enterprises-will-still-pay.html#:~:text=The%20hypervisor%20license%20is%20free,a%20list%20price%20of%20%24995. (accessed Aug. 1, 2023).

[8] S. Christensen, “How much does a small network setup cost in 2023?,” E, https://www.encomputers.com/2023/03/small-network-setup-cost/ (accessed Aug. 1, 2023).

[9] “How much does a database design service cost?,” Database Design Prices [2023]: How Much Does a Database Design Service Cost? - CostOwl.com, https://www.costowl.com/b2b/design-services/design-services-database-cost/#:~:text=Small%20business%20can%20expect%20to,the%20future%20of%20your%20company. (accessed Aug. 1, 2023).