White Paper

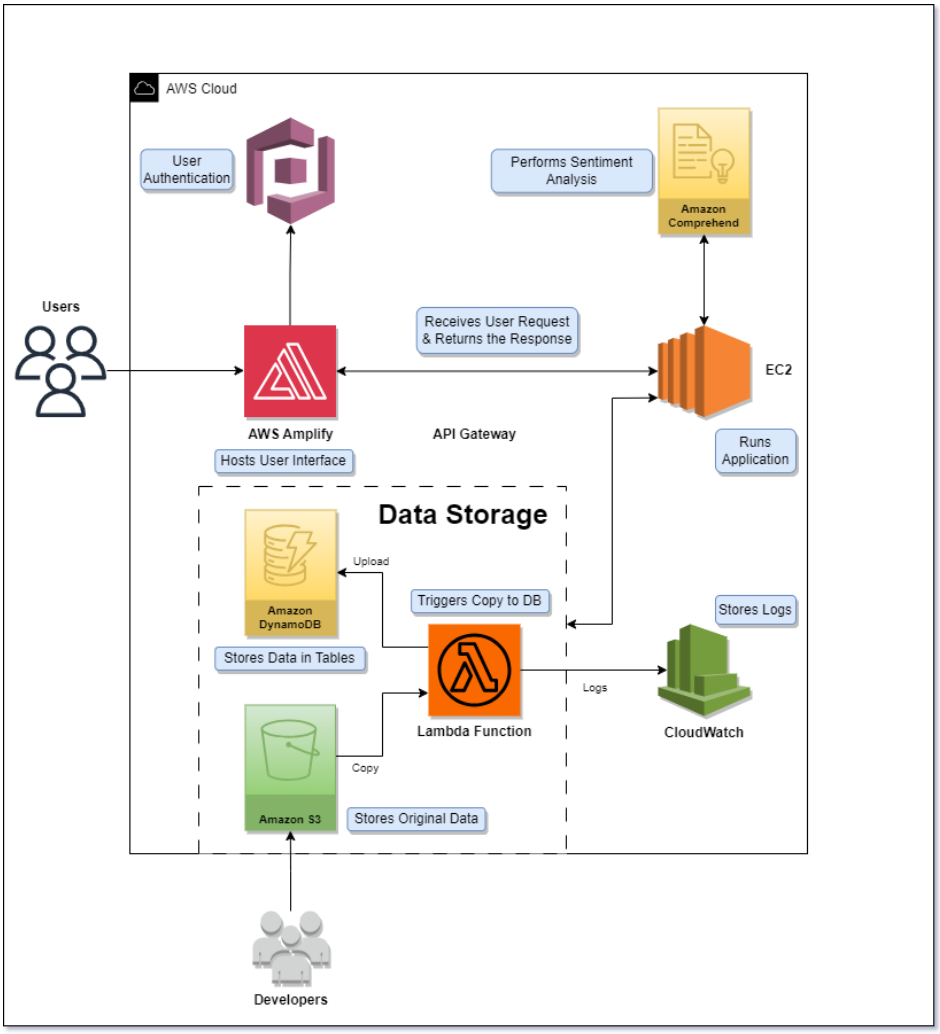
Overview:

Our project aims to analyse sentiments in Amazon product reviews and present them in an easy-to-understand format. By doing this, we help users quickly grasp how well a product is received. Additionally, our project allows users to filter sentiments based on specific date ranges, providing insights into the latest opinions about products. We also offer a subscription feature where users can subscribe to products of interest, which will be displayed in a dedicated subscription tab for convenient access. This allows users to stay updated on the latest sentiments and performance of their chosen products, enhancing their decision-making process.

Goals:

- Learn how to automate Lambda function execution when a file is uploaded to Amazon S3.
- Understand the process of populating DynamoDB tables by processing data using Lambda functions.
- Explore AWS Cognito to implement secure user authentication for your application.
- Utilize AWS Comprehend to analyse sentiments in Amazon Product Consumer Reviews.
- Learn to deploy frontend application seamlessly on AWS Amplify for efficient hosting.
- Understand the deployment process of backend applications on Amazon EC2 instances for robust backend infrastructure.
- Integrate API endpoints from your backend application deployed on EC2 with your frontend application deployed on AWS Amplify.
- Learn to use environment variables to manage configurations for different environments, ensuring flexibility and security.
- Use GitHub to fetch code directly, enabling seamless updates and reducing manual changes in infrastructure for continuous deployment.
- Master the use of Terraform to spin up all necessary AWS resources in an automated and scalable manner.

Architecture Diagram:



We have leveraged eight AWS services for our project, specifically for data processing tasks and running and operating the web application.

For data processing, we utilized following AWS services:

1. ***Amazon S3****:* We have employed an S3 bucket to upload CSV files comprising our dataset. This action serves as the trigger for our data processing flow, marking its inception and initiation.
2. ***AWS Lambda:*** The Lambda function is a critical component of our data processing workflow. It encompasses all the necessary functionalities to process CSV data and subsequently add it to the relevant tables in DynamoDB. To optimize efficiency, we implemented batch inserts and ensured data uniqueness by generating a unique Partition Key using the hashlib library. This

Lambda function is triggered automatically upon the upload of a CSV file to S3, initiating our data processing operations seamlessly.

3. ***DynamoDB:*** We use DynamoDB to store data for our application across four tables: categories, products, reviews, and subscriptions. Each table requires a unique Partition ID specified during table creation, and the uniqueness of these IDs is managed during data insertion by our Lambda function. We've opted for a Pay Per Request model over resource provisioning for Read and Write operations in DynamoDB, optimizing both read and write efficiency.
4. ***CloudWatch:*** We utilized CloudWatch to monitor our data processing workflow, providing invaluable assistance during the initial development phase by helping us identify errors in the logs and subsequently resolving them. Presently, it logs the number of records inserted into each table, offering insight into the smooth functioning of our data processing operations.

To run and operate our application, we utilized following AWS services:

1. ***Amazon Cognito:*** AWS Cognito handles user authentication by adding new users to the Cognito user pool upon signup. To authenticate, a verification code is sent to the user's registered email. Access to the application is granted only after entering this code. Cognito strictly maintains user records for authentication purposes without storing passwords.
2. ***Amazon Comprehend:*** We utilize AWS Comprehend to analyse sentiments in Amazon Consumer Product Reviews. Reviews for a requested product are fetched from DynamoDB and processed in batches by AWS Comprehend, ensuring efficient sentiment analysis. We do not store sentiment data, conducting real-time analysis instead. Sentiments are categorized as positive, negative, neutral, or mixed, along with percentages indicating the distribution of sentiments. To simplify interpretation, we treat mixed sentiments as neutral.
3. ***EC2:*** We deploy our Flask backend application on EC2, utilizing a script to automate setup and launch processes upon EC2 creation. The script fetches code from our GitHub repository's main branch, sets up a virtual environment with necessary dependencies, and assigns values

to essential environment variables. Upon successful installation of dependencies, the script starts the application using a predefined command. Additionally, an Elastic IP is allocated to the EC2 instance, enabling access to Flask application endpoints from the frontend application.

4. ***AWS Amplify:*** We utilize AWS Amplify for hosting our frontend application. We've configured necessary build settings to facilitate frontend application setup. Additionally, we've set up an environment variable using the Elastic IP assigned to EC2, enabling seamless communication between the frontend and backend components of our application.

Estimated Cost:

We predominantly rely on the AWS Free Tier for most of our application services. However, two services incurred charges: AWS Comprehend, which has a Free Tier limit of 50,000 requests, and DynamoDB, where we use a Pay Per Request model. DynamoDB charges approximately 0.25 USD per 1 million Read Requests and 1.25 USD per 1 million Write Requests. We utilized the AWS Pricing Calculator to estimate the monthly and yearly costs, as outlined below:

AWS Service	***Estimated Monthly Cost***
Amazon Simple Storage Service (S3)	0.00 USD
AWS Lambda	0.00 USD
Amazon DynamoDB	0.03 USD
Amazon CloudWatch	0.00 USD
Amazon Cognito	0.00 USD
Amazon Comprehend	10.00 USD
Amazon EC2	6.34 USD
AWS Amplify	2.35 USD

Including the AWS Free Tier, the monthly cost totals 18.72 USD, equivalent to 224.64 USD annually.

Benefits:

- Efficient processing of data using S3, Lambda, and DynamoDB, automating tasks and reducing manual effort.
- Easy integration of new categories into the data processing flow through direct addition to S3.
- Detailed logs and metrics for monitoring system performance using CloudWatch, facilitating proactive issue identification and resolution.
- Streamlining user authentication by utilizing AWS Cognito, eliminating the need for a separate user table in DynamoDB and ensuring secure user management.
- Detailed sentiment analysis of reviews using AWS Comprehend, eliminating the need for additional sentiment analysis models.
- Utilizes EC2 and AWS Amplify for seamless deployment and scaling of applications, with automatic updates from GitHub reducing manual intervention.

Lessons Learned:

Hosting applications on AWS presents challenges akin to those encountered during application development. Debugging is often necessary while setting up the required services to ensure smooth operation. Additionally, when creating Terraform scripts, it's beneficial to first manually create all necessary resources. This approach provides insights into the required resources and permissions, aiding in the development of efficient Terraform code.

What we would do differently if we had to do this over?

Given additional time, we would have prioritized creating a pipeline to fetch Amazon Consumer Reviews data and connect it to an S3 Bucket for continuous data retrieval. Furthermore, we would have embraced containerization for our application, enhancing its scalability and deployment efficiency.

Why Cloud?

Cloud infrastructure is ideally suited for our project, especially when considering scalability requirements. With minimal adjustments to the services we utilize, we can easily scale up our product as needed. In real-world scenarios, the abundance of data related to Amazon Consumer Reviews necessitates accessibility from anywhere, making cloud deployment the optimal choice. The flexibility and scalability offered by cloud services ensure seamless expansion and availability of critical data resources.

Challenges/Issues:

Only issue we encountered was during the deployment of our frontend application using AWS Amplify which required manual initiation of the build process by clicking "run build." Although we could create all necessary resources in under 2 minutes, the AWS Amplify build itself took 4 minutes thereafter. Additionally, we faced minor challenges in setting up the required permissions for our application's resources. However, through diligent debugging and troubleshooting, we successfully resolved these issues.

Other Features/Enhancements:

Given additional time, we would have endeavoured to establish a direct link between our project and actual Amazon products, providing users with a seamless redirection to the specific products. Furthermore, we would have expanded our categories and products to enhance the project's scope. Additionally, implementing functionality to group similar products together would have been a valuable addition to enhance user experience and navigation within the application.