Mealbuddy

Group Name: Group G

Group Members:

First name	Last Name	Student number
CHINJU	BABY	C0769912
ELDA	VARGHESE	C0769741
ELVIN IYPE	MATHEW	C0769974
TOM	JOSEPH	C0760915
JEENA HELEN	FRANCIS	C0764493

Submission date: ***19- AUG - 2020***

Contents

Abstract

Mealbuddy is a serverless, microservice-driven web application created completely using AWS services and third party Kommunicate API. Mealbuddy Chatbot that can simulate a conversation (or a chat) with a user in natural language. Helping the user to make wiser decisions through the recommendations made by machine learning.

Introduction

There are numerous websites and applications which can show results for restaurants and how to deliver them. But they lack the touch of personalization, where the user is given recommendations regarding what kind of food he/she might like for having it. This leads to our requirement for mealbuddy and plus the advantage of chatbot makes it more unique and friendly. The drawbacks of other apps , no suggestions from other apps makes mealbuddy the perfect project to pursue according to our current market research.

The main application of our chatbot is to provide suggestions to it's users based on the set of preferences like location, type of cuisine, number of people etc. provided to it through conversations. Mealbuddy chatbot application will send restaurant suggestions to users through SMS. We have used AWS services like Lex, Cognito, SQS, SNS, DynamoDB, CloudWatch, Sagemaker and yelp API, Kommunicate chatbot.

Project Details

Project Schedule

The project engagement began on Thu, 28 May 2020 and scheduled to end on Sun, 16 August 2020 with a total span of 12 weeks of course. The documentation was done in github along with other tasks. The weekly report requirement was also accomplished during the project weeks.

Category	Week1	Week2	Week3	Week4	Week5	Week6	Week7	Week8	Week9	Week10	Week11	Week12
Frontend												
Backend												
Chatbot												
AI												

	Design and Code Development
	Unit and Integration Testing
	Full System Testing

Project Costs

The project cost can be calculated using AWS Billing Management Tool. The total cost we tried to make it minimum but came more than we expected. The ML or Sagemaker EC2 instance endpoint cost was the major contributing factor for the expenses.

SageMaker	$45.93
SNS	$0.03
CloudWatch	$0.00
DataTransfer	$0.00
Other Services	$0.00
Tax	$0.00
Total	**$45.96**

Service	Free Tier usage limit	Current usage	Forecasted usage
Amazon Simple Storage Service	2,000 Put, Copy, Post or List Requests of Amazon S3	68 Requests	124 Requests
Amazon Simple Storage Service	20,000 Get Requests of Amazon S3	404 Requests	737 Requests
AmazonCloudWatch	1,000,000 API requests for Amazon Cloudwatch	15,945 Requests	29,076 Requests
AWS Key Management Service	20,000 free requests per month for AWS Key Management Service	88 Requests	160 Requests
AmazonCloudWatch	5 GB of Log Data Ingestion for Amazon Cloudwatch	0 GB	0 GB
AmazonCloudWatch	5 GB of Log Data Archive for Amazon Cloudwatch	0 GB-Mo	0 GB-Mo
AWS Lambda	400,000 seconds of compute time per month for AWS Lambda	165 seconds	300 seconds
Amazon Simple Storage Service	5 GB of Amazon S3 standard storage	0 GB-Mo	0 GB-Mo
AWS Lambda	1,000,000 free requests per month for AWS Lambda	310 Requests	565 Requests
Amazon Simple Queue Service	1,000,000 Requests of Amazon Simple Queue Service	233 Requests	425 Requests

Communications Strategy

The team takes advantage of different communication methods to collaborate with each member, including email, instant web meeting technology like zoom , voice calls and voicemail. The GitHub helped us developers to share and build better software. The Slack provided us with the required chat rooms for our private group and direct messaging. The notes of discussion on meetings and other conversation about the project is stored in both email and google drive.We utilized regular meetings to solicit feedback for improvements and changes.

SDLC Strategy

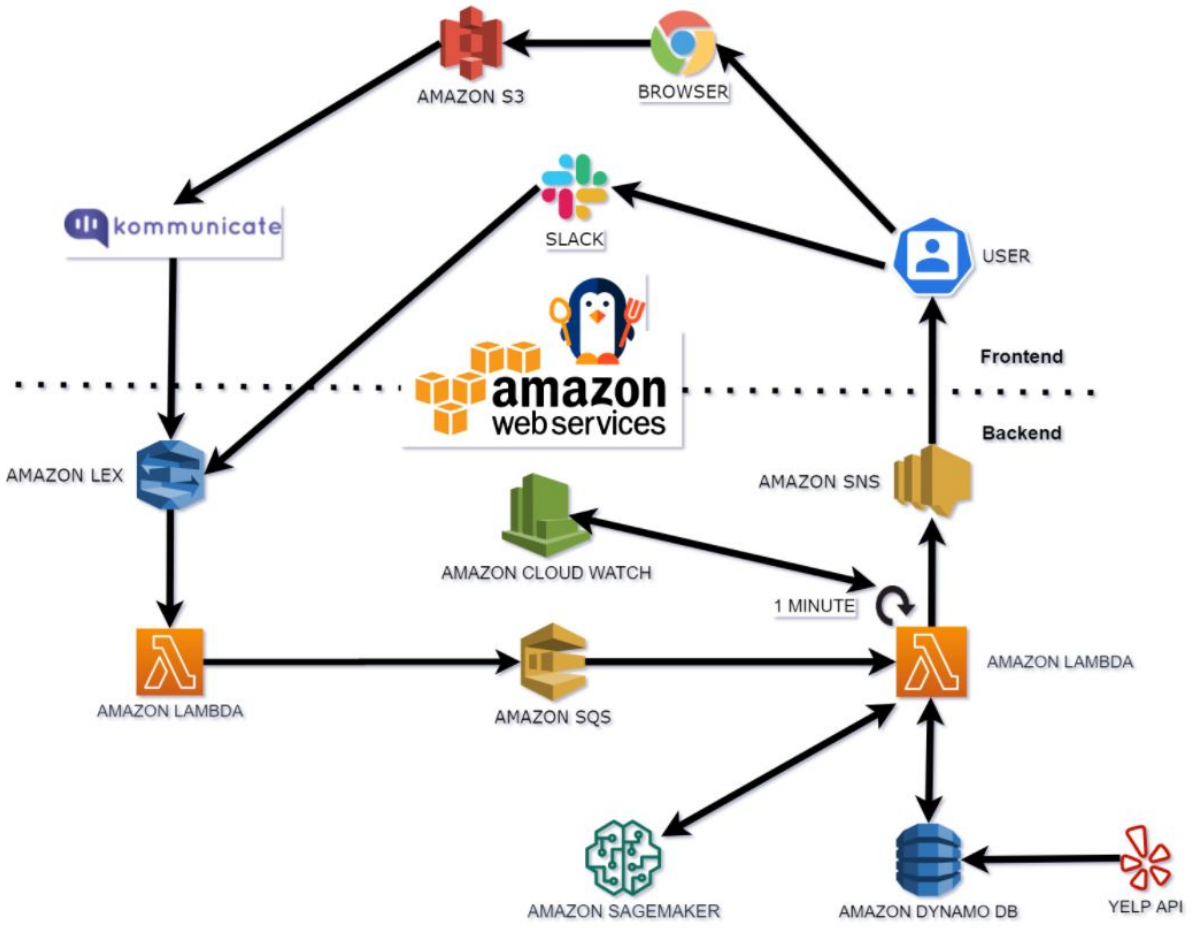
Jira is the leading project management software across multiple industries today, and for good reason. Jira allows you to view, track, and report on tasks you're currently working on, while providing full transparency to management in terms of what is being accomplished on a large scale.

Project Limitations

- The restaurant is limited to the Toronto, Canada region or area.
- The recommendation is currently based on factors like cuisine, ratings, review countings.

Architecture Diagram

The user interacts with chatbot in Amazon Lex through any browser or through Slack App. The browser fetches the web application from the Amazon S3. The website contains the Kommunicate widget for interaction with the Amazon Lex. Also the Slack App is also connected to Amazon Lex through Oauth key channels.



The Amazon Lex is connected with two Amazon Lambda functions LF1 and LF2. LF1 is used to perform the chat operations. Then the collected information is pushed from the user to Amazon SQS queue. LF2 will pull the message from the sqs queue and process it. Amazon DynamoDB collects or scrapes the full details of restaurants from the YELP API. The collected restaurant details are fed to the Amazon Sagemaker Endpoint which will suggest 3 restaurants and the reply is sent to the users through the sms using the Amazon SNS. Also the request message and the reply message is stored in another table in DynamoDB.

The various services used during the development process are

S3

Simple Storage Service, is one of Amazon's longest running AWS offerings. It's purpose is to provide easy-to-use cloud storage that is scalable, highly available, and reliable.

We hosted our web application in S3 bucket.

Lambda

AWS Lambda runs code called a Lambda function in response to an event. An event could be anything from a user uploading an image into an S3 bucket to a user tapping a button on your mobile app to buy an item. All you have to do is upload your code, and AWS Lambda automatically runs your code and scales your application for you. You only pay for the time your code spends running and each event trigger, which helps you keep your cost low. There are no servers to provision or manage and nothing to scale, because Lambda takes care of all that for you, so you can focus on writing code. You can have a Lambda function up and running with just a few clicks, saving you hours and hours of backend provisioning work.The code we run on Lambda is known as a function. In mealbuddy we have created three lambda functions namely LF1 ,LF2 , ScrapeYelpData.

Lambda function named 'LF1' handles three main intents namely greeting intent,dining suggestion intent and thank you intent from Lex. Input json data (like phone number) validation is done in this function.

Lambda function 'LF2 'acts as a queue worker. Whenever it is invoked it pulls a message from the SQS queue 'LF1SQSLF2'. Lambda pulls the message using the function sqs.receive_message and retrieves all the parameters.

ScrapeYelpData Lambda function insert yelp API restaurant details to DynamoDB table called yelp_restaurant. Users and applications can access the lambda and its functions and layers by enabling the required policies related to dynamoDB and other AWS services.

SQS

It is called Service Broker. It could actually break messages between services so that one service could send a message to the Service Broker, and then another service could come . Rather than the user having to manage these queues of messages, users could just have a prebuilt service that did it for them.This service happens to be called the Simple Queue Service, and it can be found in the Application Integration category. So it really is all about getting our applications to integrate, to work together, and that's what the Simple Queue Service, or SQS, is all about.
.This service enables the users to manage the messages efficiently. The amount of traffic SQS can scale up or down without any configuration.

Created fifo sqs 'LF1SQSLF2.fifo' to receive messages from lambda 'LF1 '. Added 'All sqs action' permission to the created sqs to pull or pop the messages. We have also added dead letter sqs 'deadSQS.fifo' for isolated messages that cannot be processed correctly.

Lex

Lex allows you to create bots that can serve multiple intentions . This means that you can use lex to power facebook messenger bots, add functionality to Slack , or even send messages to your users.

Amazon Lex follows the basic chatbot architecture. It's basically a service for making CUI for many applications with the utilizing voice and text. We need to create intents, entities, your script and choose the dialog flow (user story)

- Intents - Intents is like a verb, that is used for detecting the intention of the user. Intent is like a person sitting in front desk of a large building , so that the intention of anyone walking to the building can be easily understood.
- Entities - If Intents are verb, entities are like a user noun. It's also called as slots. Some examples are date, time, cities, names, brands etc. We need to capture these entities, it's a very important part as it needs to take action based on the user intents.
- mealbuddy app with a bot that needs to determine if you're ordering an indian cuisine,chinese cuisine or Italian cuisine. Based on which one you want to order, the bot needs to gather specific data. So it has different questions for each path, or different intentions of the user. Once all the missing pieces are filled in, it can fulfill the request. Fulfilling the request means doing what it was designed to do once it's gathered all the data. That will vary depending on the design of the bot, and the purpose of our app.

Yelp Fusion Api

Yelp Api provides the best local contents and the user reviews ,rating and all details like type of cuisine, location of the restaurants. There are two types of API's available on Api platform, Fusion and GraphQL. Yelp Fusion API is a combinationation of multiple api's which are combined into a single package, so we can make calls directly from the browser.

- First we create an account in YELP API and enter some details about the app
- Then we got an client id and API key

After generating an access token, we can pull the information from the yelp.

DynamoDB

DynamoDB helps to build our prototype on DynamoDB tables and as our app grows we can go from zero to let's say millions of read and write requests without ever having to worry about provisioning servers, clustering, upgrades, patching, and replication. All these issues are handled by AWS DynamoDB. There are mainly three core components for dynamoDb namely items ,attributes and tables. Group of attributes is called an item and similarly a group of items is called tables.

We have created two tables yelp_restaurant for storing restaurant details and mealbuddy_customer_responses for storing customer responses and error messages. The yelp_restaurant table stores data in different attributes like id, address, alias, coordinates , cuisine, name, rating , review_count etc.

SNS

Amazon Simple Notification Service (SNS) is a fully managed messaging service for both system-to-system and app-to-person (A2P) communication.[8] It enables you to communicate

between systems through publish/subscribe (pub/sub) patterns that enable messaging between decoupled microservice applications or to communicate directly to users via SMS, mobile push and email.

In our project we have created a topic 'MealbuddySNS' for sending restaurant suggestion SMS to the user and the event is triggered by AWS lambda LF2 using the Cloudwatch scheduled for 1 minute interval.

Kommunicate

Kommunicate is a third party chatbot widget. Kommunicate provides code less integration with AWS Lex ,Google Dialogflow etc. Once integrated, users can chat with the bot using a beautiful and customizable chat-widget.(Human+bot hybrid customer support platform)

We have integrated the Kommunicate widget to the webpage stored in S3.

Sagemaker

AWS SageMaker is the root of all machine learning, or ML solutions, deployed through AWS. SageMaker equips each piece of the stack of requirements for a machine learning solution. All machine learning solutions require articulated goals, data, algorithms that set the process of how to learn, and the deployment of the results of the learning. SageMaker supplies each of these pieces of the puzzle. Their supply pieces are also available for adjustment if the solution provider needs a different learning set or organizational goal.Advantage of using sagemaker is that it provides integrated Jupyter authoring notebook instances. It also provides most of the complicated ML algorithms.

In our project we have trained and tested random forest and xgboost ML algorithms with the help of sage maker and we could find out the accuracy of different models. Also we deployed the XGBoost model to the sagemaker endpoint to be called from LF2.

Slack

Slack is a messaging platform where people can communicate and work together with their team members and connect their software tools and services. slack gives an organized conversation by creating channels where the team members can share their files and send messages. we can add apps to our workspace and use the apps without leaving the workspace to connect with the services and tools.[5]

We have integrated our application mealbuddy to slack, so the users can efficiently use the slack for getting the best recommendation of restaurants.

CloudWatch

With the help of Amazon Cloud Watch, we can monitor real time Amazon Web Service resources and applications. The home page of cloud watch defaultly shows the metrics of different amazon services we are using currently. Moreover we can create an alarm to check the metrics and to send notifications if the threshold point is reached.

We have created an alarm to trigger the LF2 function to process and send the reply message as SMS.

Project Procedure

1. Build and deploy the frontend of the application

Implement a user interface, where the user can write messages and get responses back.
Host the website in AWS S3 bucket
Step 1: Create a bucket named www.mealbudy.xyz
Step 2: Enable static website hosting
Step 3: Edit block public access settings
Step 4: Add a bucket policy that makes your bucket content publicly available
Step 5: Configure an index document
Step 6: Test your website endpoint

2. Build a mealbuddy chatbot using AWS Lex

a. Created a new bot mealbuddy using amazon Lex service
b. Implement the following three intents:
   i. GreetingIntent
   ii. ThankYouIntent
   iii. SuggestDiningIntent
c. The implementation of an intent entails its setup in Amazon Lex as well as handling its response in the Lambda function code hook. Example: for the GreetingIntent you need to
   i. Create the intent in Lex,
   ii. Train and test the intent in the Lex console,
   iii. Implement the handler for the GreetingIntent in the Lambda code hook, such that when you receive a request for the GreetingIntent you compose a response such as “Hi there, how can I help?”

3. Integrate the Lex chatbot using Kommunicate

Using Kommunicate, which provides a code-less integration with Amazon Lex.[4]

Step 1: Create a free Kommunicate account - You can create a free account in Kommunicate. Head to the signup section to start.

Step 2: Connect your Amazon Lex bot - Post signup, navigate to the bot integration section and select the Amazon Lex platform. Kommunicate requires the below detail in order to query your bot on your behalf. You just need to fill a few details to connect your Lex bot. You can get these details in your AWS Management Console -> Security credentials section.

- Access key ID & Secret access key
- Bot name in Lex platform
- Bot alias
- Region

Once you have the above information follow the below steps, click Save and Proceed.

Step 3: Give your bot an identity - You can give your bot a name and a profile picture. The name and the profile picture will be visible to your users while they interact with your bot. Give your bot a name. This name will be visible to your users who interact with your bot. Click Save and Proceed.

Step 4: Enable/Disable human handoff - Your bot is as smart as you can make it. But at times, it may fail to understand the questions of a user. In that case, you can trigger a chatbot to human handoff. This helps you make the overall user experience better and handle edge cases. Choose whether to enable or disable this feature and click on Finish bot integration setup.

Step 5: Assign all the incoming conversations to your Lex bot - To let your user chat with the new bot, you need to assign all the conversation to the bot. Just after finishing the bot setup, click on Let this bot handle all the incoming conversations. Now, all new Conversation initiated after the integration will be assigned to this bot and your bot will start answering them.

Step 6: Install the Kommunicate on your website

4. Integrate Slack to Lex

Slack is a proprietary business communication platform developed by American software company Slack Technologies. Slack offers many IRC-style features, including persistent chat rooms organized by topic, private groups, and direct messaging.

Step 1: Create an Amazon Lex Bot

Step 2: Sign Up for Slack and Create a Slack Team - Sign up for a Slack account and create a Slack team.

Step 3: Create a Slack Application

a. Create a Slack application on the Slack API Console
b. Configure the application to add interactive messaging to your bot: you will get application credentials (Client Id, Client Secret, and Verification Token).

Step 4: Integrate the Slack Application with the Amazon Lex Bot

- Sign in to the AWS Management Console, and open the Amazon Lex console at https://console.aws.amazon.com/lex/.
- Choose the Amazon Lex bot that you created in Step 1.
- Choose the Channels tab.
- In the left menu, choose Slack.
- On the Slack page, provide the following:
- Type a name. For example, BotSlackIntegration.
- Choose "aws/lex" from the KMS key drop-down.
- For Alias, choose the bot alias.
- Type the Client Id, Client secret, and Verification Token, which you recorded in the preceding step. These are the credentials of the Slack application.
- Choose Activate. The console creates the bot channel association and returns two URLs (Postback URL and OAuth URL). Record them. In the next section, you update your Slack application configuration to use these endpoints as follows:
- The Postback URL is the Amazon Lex bot's endpoint that listens to Slack events. You use this URL:
- As the request URL in the Event Subscriptions feature of the Slack application.
- To replace the placeholder value for the request URL in the Interactive Messages feature of the Slack application.
- The OAuth URL is your Amazon Lex bot's endpoint for an OAuth handshake with Slack.

Step 5: Complete Slack Integration

- Update the OAuth & Permissions feature as follows:
    - In the left menu, choose OAuth & Permissions.

- In the Redirect URLs section, add the OAuth URL that Amazon Lex provided in the preceding step. Choose Add a new Redirect URL, and then choose Save URLs.
- In the Bot Token Scopes section, add two permissions with the Add an OAuth Scope button. Filter the list with the following text:
  - chat:write
  - team:read

- Update the Interactivity & Shortcuts feature by updating the Request URL value to the Postback URL that Amazon Lex provided in the preceding step. Enter the postback URL that you saved in step 4, and then choose Save Changes.
- Subscribe to the Event Subscriptions feature as follows:
  - Enable events by choosing the On option.
  - Set the Request URL value to the Postback URL that Amazon Lex provided in the preceding step.
  - In the Subscribe to Bot Events section, subscribe to the message.im bot event to enable direct messaging between the end user and the Slack bot.
  - Save the changes.

Step 6: Test the Integration

- Choose Manage Distribution under Settings. Choose Add to Slack to install the application. Authorize the bot to respond to messages.
- You are redirected to your Slack team. In the left menu, in the Direct Messages section, choose your bot. If you don't see your bot, choose the plus icon (+) next to Direct Messages to search for it.
- Engage in a chat with your Slack application, which is linked to the Amazon Lex bot. Your bot now responds to messages.

5. Create a Lambda function using AWS Lambda service(LF1)

Create a Lambda function (LF1) and use it as a code hook for Lex, which essentially entails the invocation of your Lambda before Lex responds to any of your requests (handles three main intents namely greeting intent,dining suggestion intent and thank you intent from Lex.)

a. For the DiningSuggestionsIntent, you need to collect at least the following pieces of information from the user, through conversation : Location, Cuisine, Dining Time, Number of people, Phone number
b. Added validation for the input json data (like phone number) in the python code.
c. Tested the intents with three different test events (different json input).
d. and test it in the Amazon Lex console.

6. Push the information to an SQS queue

Based on the parameters collected from the user,(through mealbuddy chatbot) push the information collected from the user (location, cuisine, etc.) to an SQS queue .

- Created fifo sqs 'LF1SQSLF2.fifo' to receive messages from lambda 'LF1 '.FIFO sqs strictly maintain the order in which the messages arrive while the default standard sqs occasionally deliver messages in order.
- Added dead letter sqs 'deadSQS.fifo' for isolated messages that cannot be processed correctly.

Also confirm to the user that we received their request and that we will notify them over SMS once you have the list of restaurant suggestions.

7. Insert into DynamoDB : Use the Yelp API to collect 5,000+ random restaurants from Toronto

- Get restaurants by your self-defined cuisine types
- You can do this by adding cuisine type in the search term (ex. Term: chinese restaurants)
- Each cuisine type should have 1,000 restaurants or so.

8. Create a lambda function to insert yelp API restaurant details to DynamoDB

- Created scrapeYelpData Lambda function to insert yelp API- restaurant details to DynamoDB table called 'yelp_restaurant'
- https://github.com/tom5167/mealbuddy/blob/master/LAMBDA/scrapeYelpData/lambda_function.py

9. DynamoDB (a noSQL database)

- Create a DynamoDB table and named "yelp-restaurant"
- Store the restaurants you scrape, in DynamoDB (one thing you will notice is that some restaurants might have more or less fields than others, which makes DynamoDB ideal for storing this data)
- With each item you store, make sure to attach a key to the object named "id" with the value of the time and date of when you inserted the particular record

- Store those that are necessary for your recommendation.(Requirements: Business ID, Name, Address, Coordinates, Number of Reviews, Rating, Zip Code)
- Created a second table called mealbuddy_customer_responses for storing customer responses and error messages.

10. Create Sagemaker Endpoint

Make a restaurant suggestion system based on reviews and ratings by using Amazon Sagemaker Service

1. Create a Notebook instance - resturantRecommendationML

- Notebook instance - typeml.t2.medium
- Volume Size - 5GB EBS

2.Gathering Data

- Gathered data using yelp api and dump to yelp_restaurant.csv

3.Data Preparation

- read yelp_restaurant.csv and store in data frame
- add column 'recommended'
    - 1 based on rating > avg(rating) and review_counting > avg(reviews_counting)
    - 0 based on rating < avg(rating) and review_counting < avg(reviews_counting)
- create df_recommend and sort by rating and reviews_counting
- create df_non_recommend and sort by rating and reviews_counting
- create semisupervised_train will 80% df_recommend + 80% df_non_recommend and save as 'semisupervised_train.csv'
- create unsupervised_train will 20% df_recommend + 20% df_non_recommend and save as 'unsupervised_train.csv'

- Convert categorical variable into dummy/indicator variables for cuisine

ex.

review_count rating cuisine_american cuisine_chinese cuisine_greek cuisine_indian cuisine_italian cuisine_latin cuisine_mexican cuisine_persian cuisine_spanish

98 4.5 0 0 0 1 0 0 0 0 0

- train, validation = train_test_split(semisupervised_train_for_model, test_size=0.1, random_state=42)
- train.to_csv('train.csv', index=False, header=False)
- validation.to_csv('validation.csv', index=False, header=False)

4.Choosing a Model and Parameter Tuning

- trainDF, validationDF = train_test_split(trainingDF, test_size=0.1, random_state=42)
- decison tree > random forest > xgboost algorithm
- xgb = sagemaker.estimator.Estimator(container,role,train_instance_count=1,train_instance_type='ml.m4.xlarge', output_path='s3://{}/output'.format(bucket),sagemaker_session=sess)
- xgb.set_hyperparameters(max_depth=3,eta=0.2,gamma=4,min_child_weight=4, subsample=0.8,silent=0, objective='binary:logistic',num_round=20)
- xgb.fit({'train': s3_input_train,'validation': s3_input_validation})

5.Deploy

- xgb_predictor = xgb.deploy(initial_instance_count=1,instance_type='ml.m4.xlarge')

6.Prediction

- xgb_predictor.predict(array).decode('utf-8')

11. Create a new Lambda function (LF2) that acts as a queue worker

The LF2 is crucial to the application and handles the majority of processes.

- Read message from SQS - if empty send error message
- Delete the processed message using receipt handle
- Search DynamoDB for restaurant details based on user input (Max 10)
- Using Sagemaker Endpoint call make prediction and Top 3 results
- Format the reply for user
- Send the message using SNS
- If any error send error message
- Store the request and reply message in DynamoDB

12. Scheduler Queue worker

We set up a CloudWatch event trigger that runs every minute and invokes the Lambda function as a result:

https://docs.aws.amazon.com/AmazonCloudWatch/latest/events/RunLambdaSchedule.html

This automates the queue worker Lambda to poll and process suggestion requests on its own.

13. SNS(Simple Notification Service)

LF2 sends text messages to the phone number included in the SQS message, using SNS.

Results

The mealbuddy application was able to respond to user queries and give recommendations to users through sms.

The mealbuddy chatbot application built is a good example of serverless micro services based web application. AWS provides a good partner for creating such applications on cloud. The services provided by amazon are pretty good and deliver the services upto the expectations. We also explored the AWS big data and machine learning services such as AWS sagemaker.

Conclusions and Future Work

The lack of experience with cloud services made our work a bit difficult but the useful AWS documentation made it seamless and easy. The documentation by AWS is apt and resourceful. Having used these services helped us to understand the power of cloud computing. AWS Free Tier allowed us to gain first-hand experience with AWS services for free. Also they provide access to up to 60 products for building on the AWS platform.

The future development is to make the application more secure and fault tolerant. To achieve fault tolerant in future we plan to add an additional AWS SQS queue and add an additional load balancer. Also will plan to add api gateway for api access for other applications through the web. The email services will also be used along with the SMS provided by AWS SNS.

Acknowledgement

We would like to express my very great appreciation to faculty supervisor Mr. William Pourmajidi for his valuable and constructive suggestions during the planning and development of this research work. His willingness to give his time so generously has been very much appreciated.

References

[1] https://docs.aws.amazon.com/AmazonS3/latest/dev/WebsiteHosting.html

[2] https://docs.aws.amazon.com/AmazonS3/latest/dev/HostingWebsiteOnS3Setup.html

[3] https://docs.aws.amazon.com/lex/latest/dg/using-lambda.html

[4] https://www.kommunicate.io/blog/amazon-lex-tutorial/

[5] https://slack.com/intl/en-ca/help/articles/115004071768-What-is-Slack-

[6] https://github.com/tom5167/mealbuddy

[7]https://aws.amazon.com/s3/

[8]https://aws.amazon.com/sns/?whats-new-cards.sort-by=item.additionalFields.postDateTime&whats-new-cards.sort-order=desc