Documentation for Assignment 5

Part A. Explore & Build a Use Case:

AWS Kinesis[1] is used to collect and process a large stream of data in real-time processing. Kinesis Video stream and data stream does real-time data and video processing, respectively.

As the Kinesis process real-time data, we can use this service for object detection in the live surveillance system. Let us suppose there is a live surveillance system running in the company and its one of the functions is to detect if the person is wearing the assigned dress code or not. Then AWS Kinesis can process the real-time data, which then goes through some analysis and finally an algorithm to detect the object.

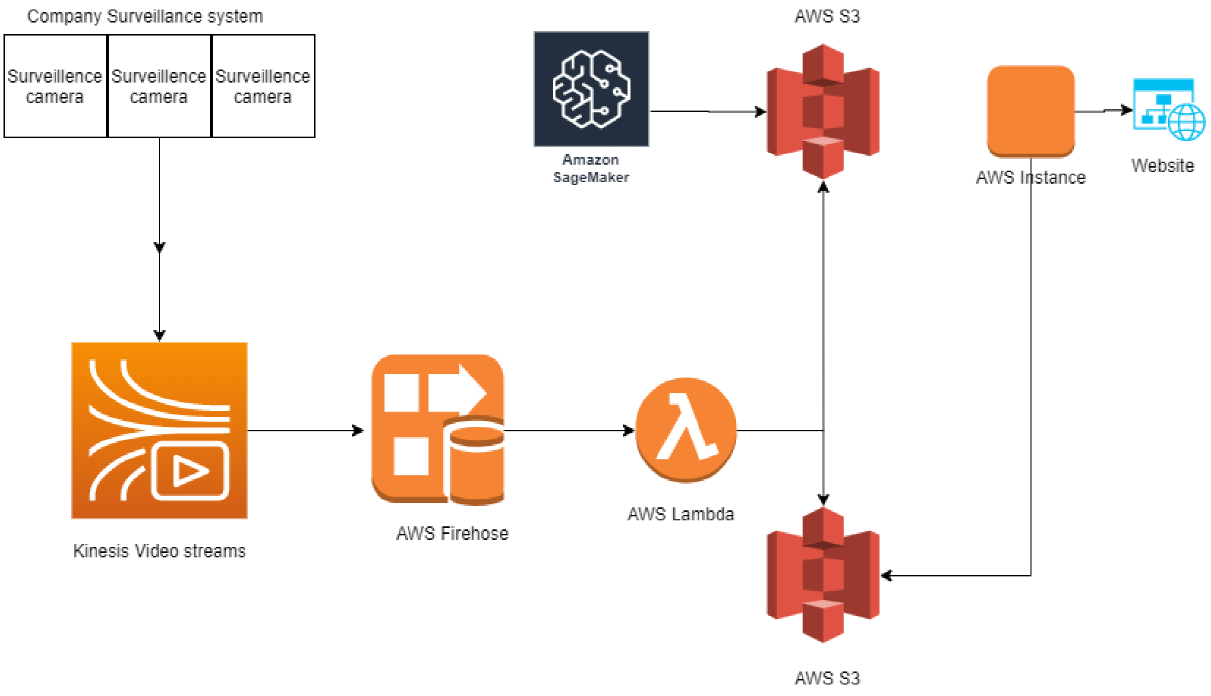
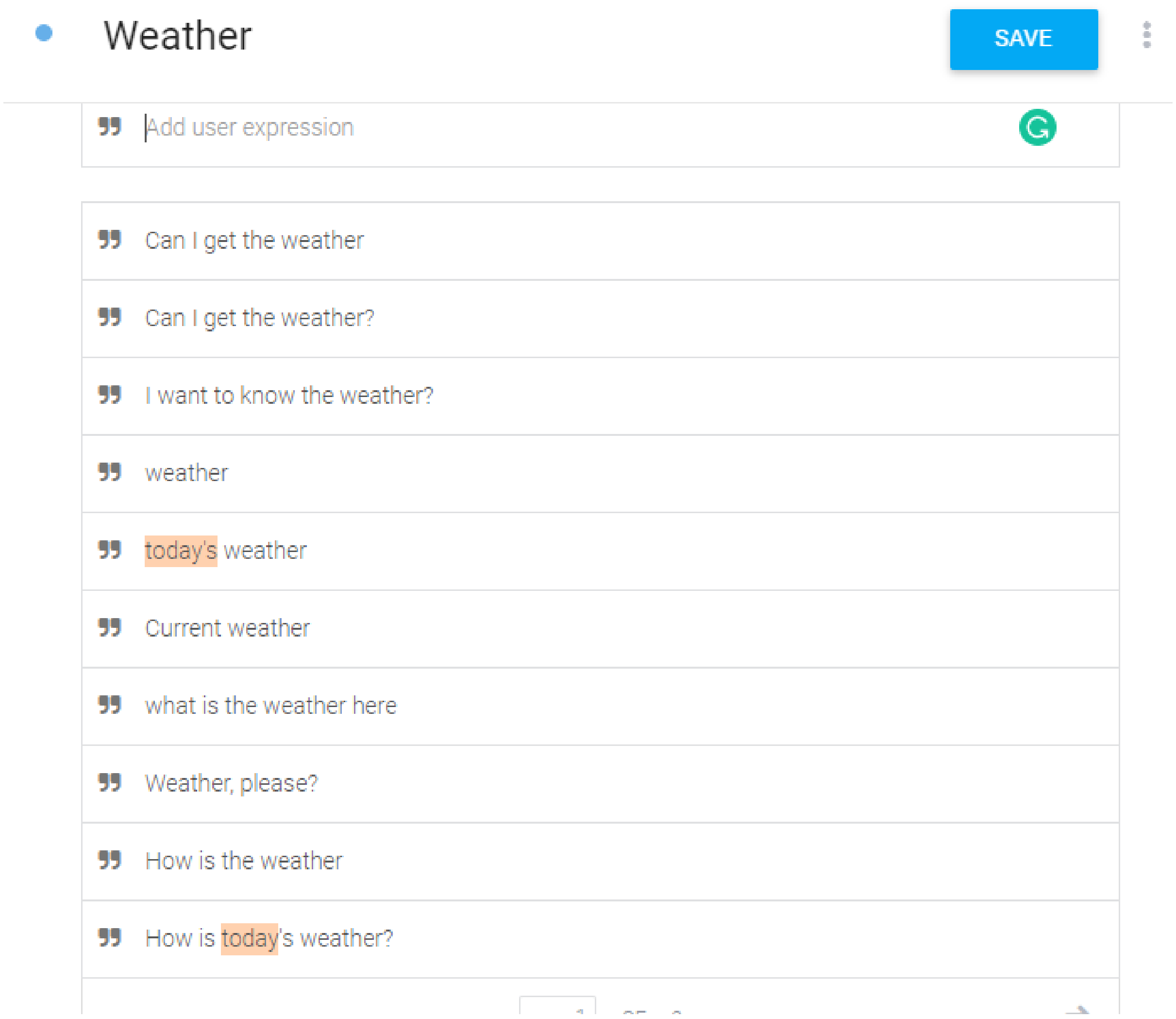


Figure a use case

In the use case diagram of the system briefly described above, here are the steps being followed:

1. Kinesis Video streams fetch real-time data from the company surveillance system.
2. The fetched data passes to AWS Firehose where ETL processing is performed and data is cleaned. Only the required data is moved further.
3. After that, AWS lambda is triggered which fetches the model from AWS S3, apply the prediction of the ML algorithm on the data from AWS Firehose, and store the prediction in another S3 bucket.
4. AWS Sagemaker is used to train the ML algorithm for object detection. All the training samples will be fetched from the S3 bucket and trained algorithms will again be stored in the same bucket.
5. Finally, the website which is hosted on AWS instance, will fetch the latest data from AWS S3 and preview it on the website.

Part B: Use of GCP AI:



Can I get the weather

Can I get the weather?

I want to know the weather?

weather

today's weather

Current weather

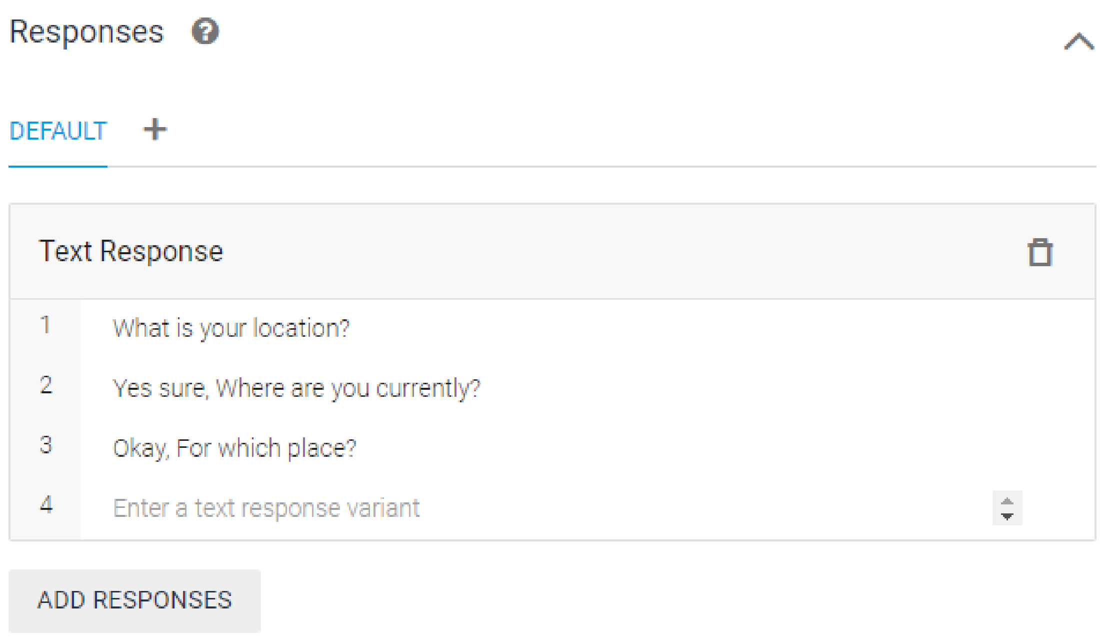
what is the weather here

Weather, please?

How is the weather

How is today's weather?

Above image shows weather questions


Responses

DEFAULT +

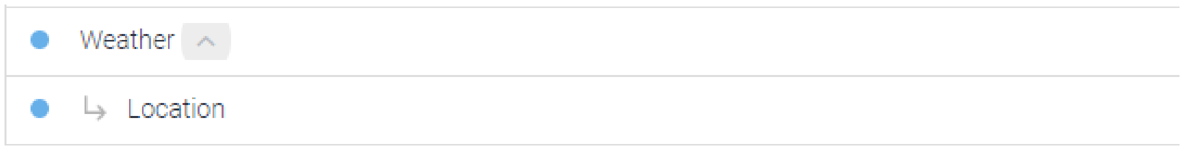
Text Response

1	What is your location?
2	Yes sure, Where are you currently?
3	Okay, For which place?
4	Enter a text response variant

ADD RESPONSES

Figure b Responses questions

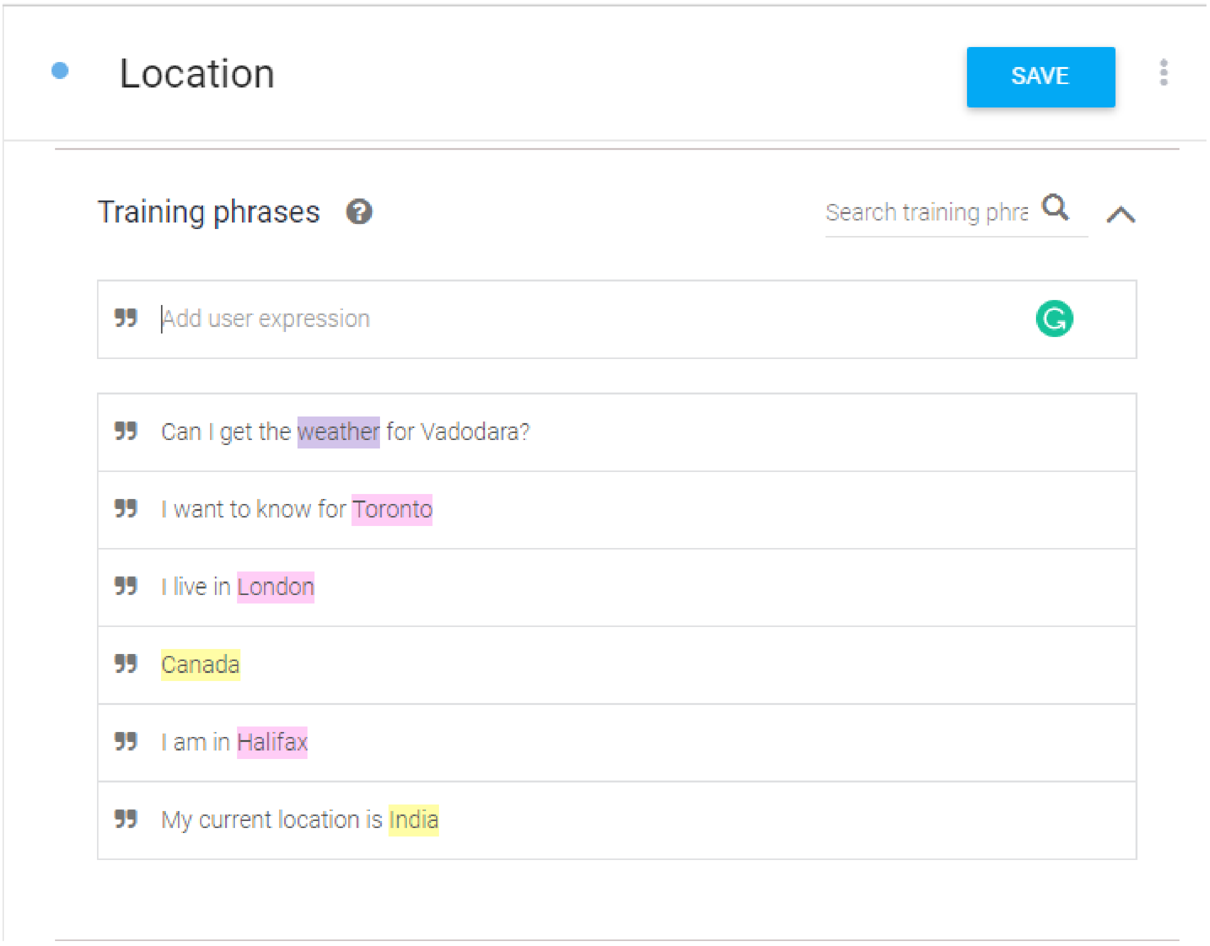
Above image shows the questions of the responses.


● Weather

● ↳ Location

Figure c follow up intent

A new intent ’Location’ is created as a follow up intent of Weather.


Location

SAVE

Training phrases

Search training phra

Add user expression

Can I get the weather for Vadodara?

I want to know for Toronto

I live in London

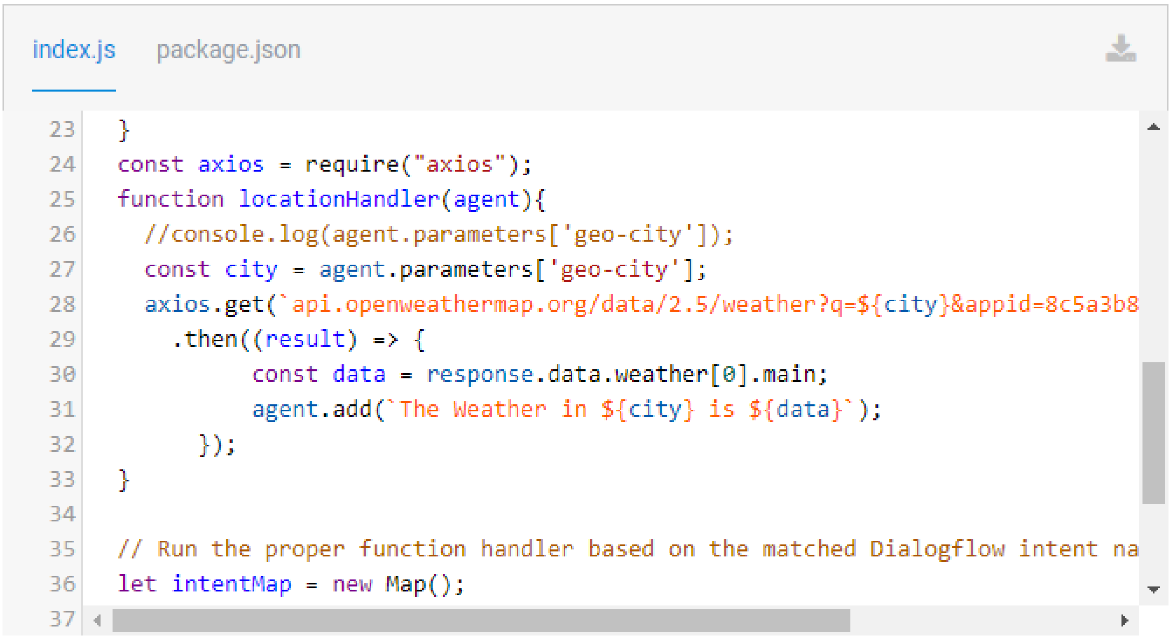
Canada

I am in Halifax

My current location is India

Figure d Location training phrases

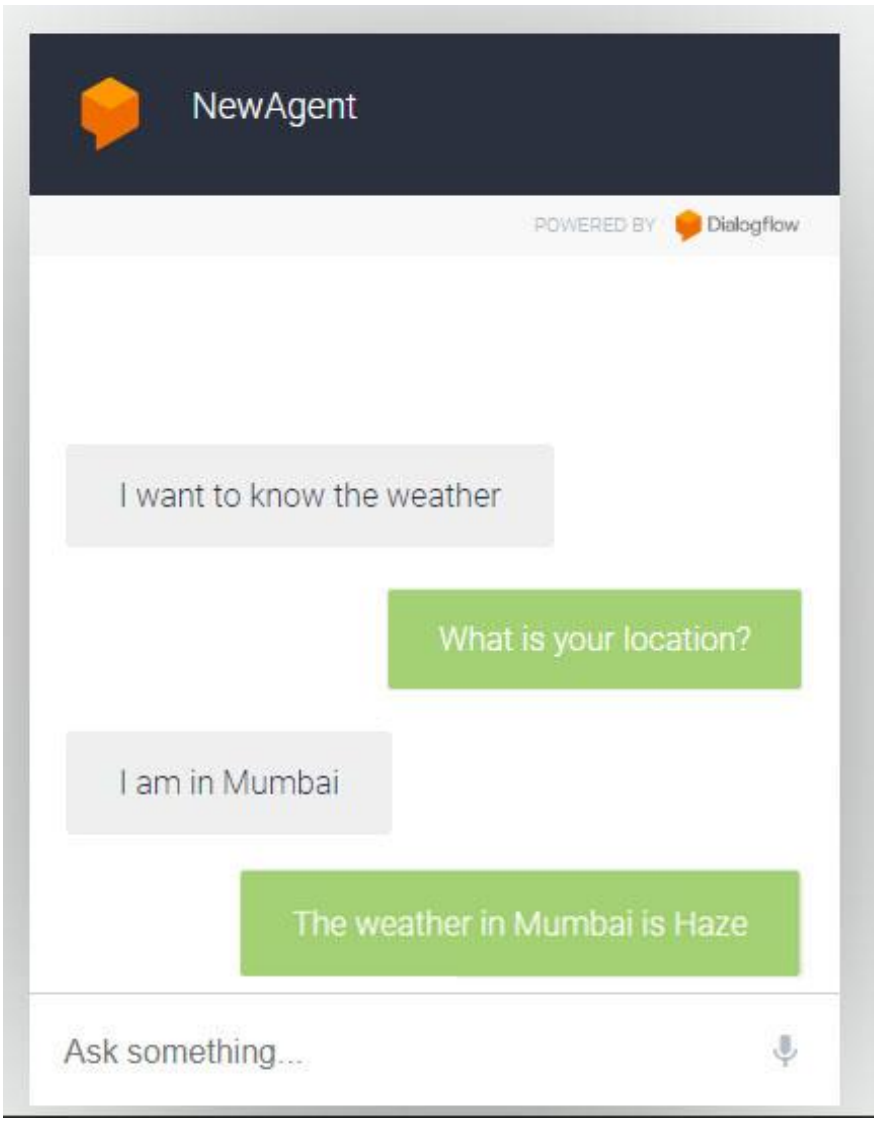
Above image shows the phrases for the Location intent.


index.js package.json

```
}
const axios = require("axios");
function locationHandler(agent){
  //console.log(agent.parameters['geo-city']);
  const city = agent.parameters['geo-city'];
  axios.get(`api.openweathermap.org/data/2.5/weather?q=${city}&appid=8c5a3b8
    .then((result) => {
          const data = response.data.weather[0].main;
          agent.add(`The Weather in ${city} is ${data}`);
        });
}

// Run the proper function handler based on the matched Dialogflow intent na
let intentMap = new Map();
```

Figure e Fulfillment code [2]

Finally above image shows the response for the Location intent.



NewAgent

POWERED BY Dialogflow

I want to know the weather

What is your location?

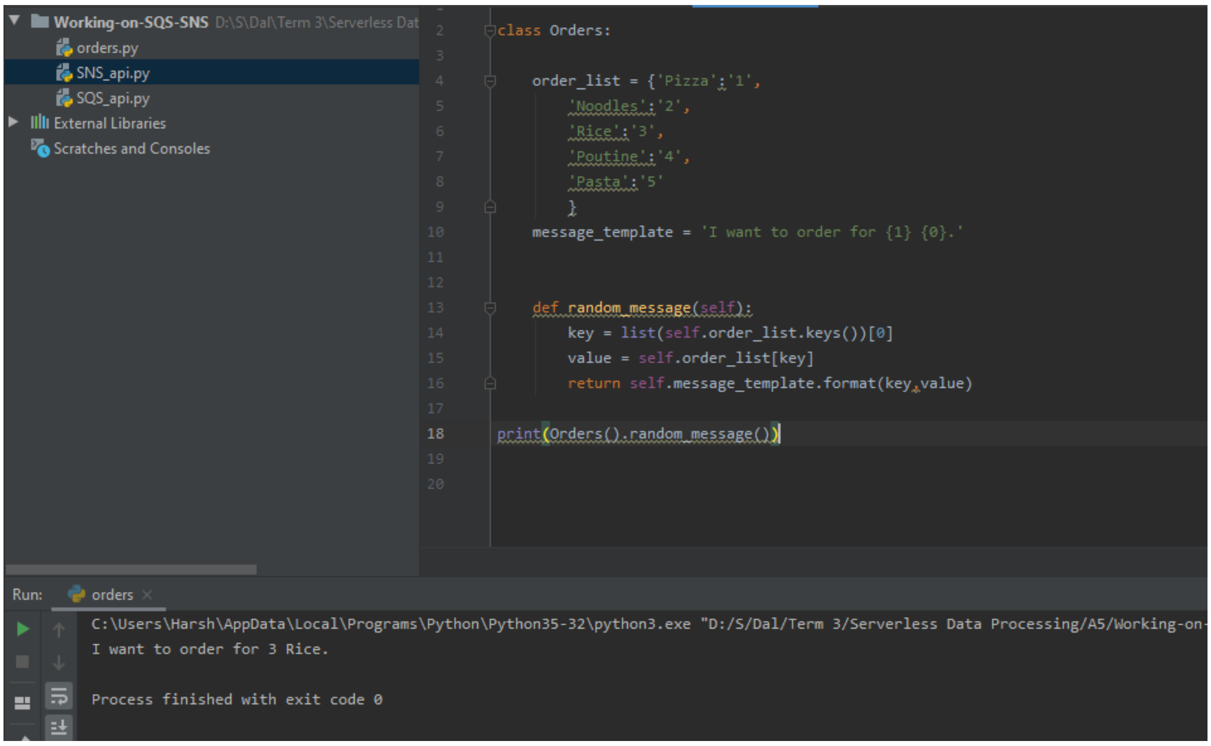
I am in Mumbai

The weather in Mumbai is Haze

Ask something...

First we create an intent in the dialogflow. Intents have entities that are fetched from the user queries and accordingly, responses are given. For an instance, in this weather chatbot, if a user asks “I want to know the weather of Halifax?”, then “weather” and “Halifax” are the entities. We first create some question samples to train the dialogflow. Many questions that the user can ask in different ways are provided. Then I created a follow up intent which means that it will be triggered after the previous one. The chatbot then asks the user for the location in this ‘Location’ intent and finally calls the fulfilment function after user responds with the location. The location is provided in the weather api, which gives the weather of the city. The result is added to the agent (chatbot) and shown on the chatbot.

Part C. Use AWS Lambda-SQS-SNS:

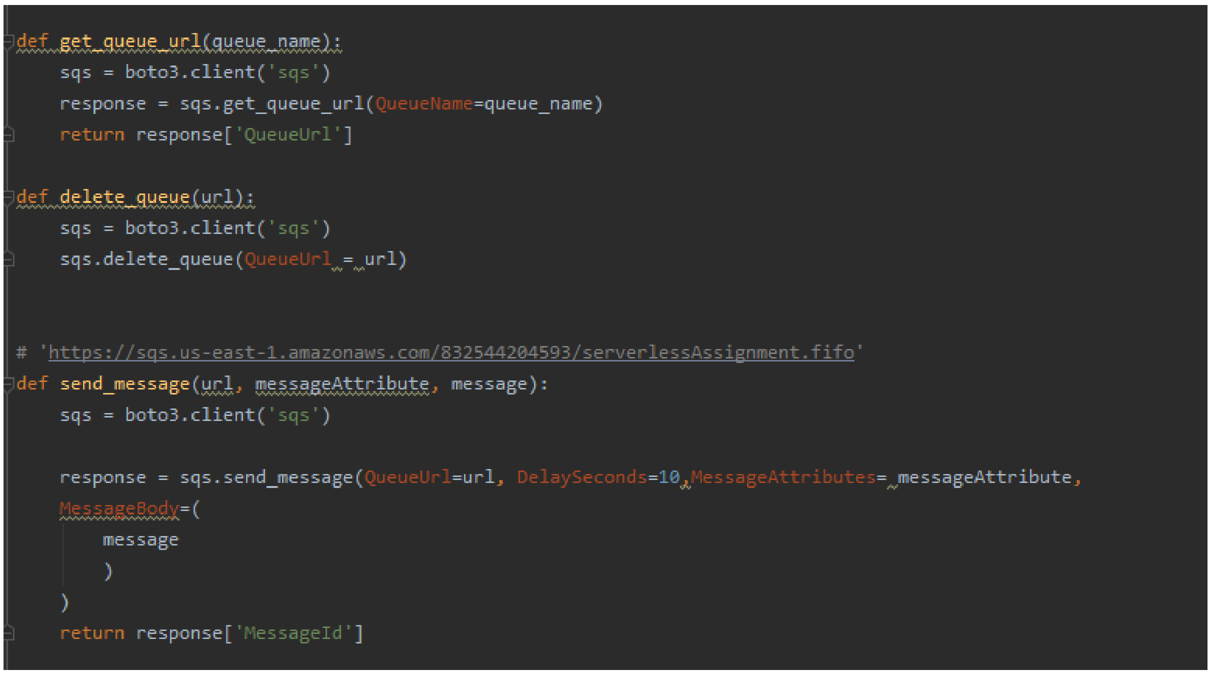


```python
class Orders:

    order_list = {'Pizza':'1',
        'Noodles':'2',
        'Rice':'3',
        'Poutine':'4',
        'Pasta':'5'
        }
    message_template = 'I want to order for {1} {0}.'


    def random_message(self):
        key = list(self.order_list.keys())[0]
        value = self.order_list[key]
        return self.message_template.format(key,value)

print(Orders().random_message())
```

```
C:\Users\Harsh\AppData\Local\Programs\Python\Python35-32\python3.exe "D:/S/Dal/Term 3/Serverless Data Processing/A5/Working-on-
I want to order for 3 Rice.

Process finished with exit code 0
```

Figure Creating orders

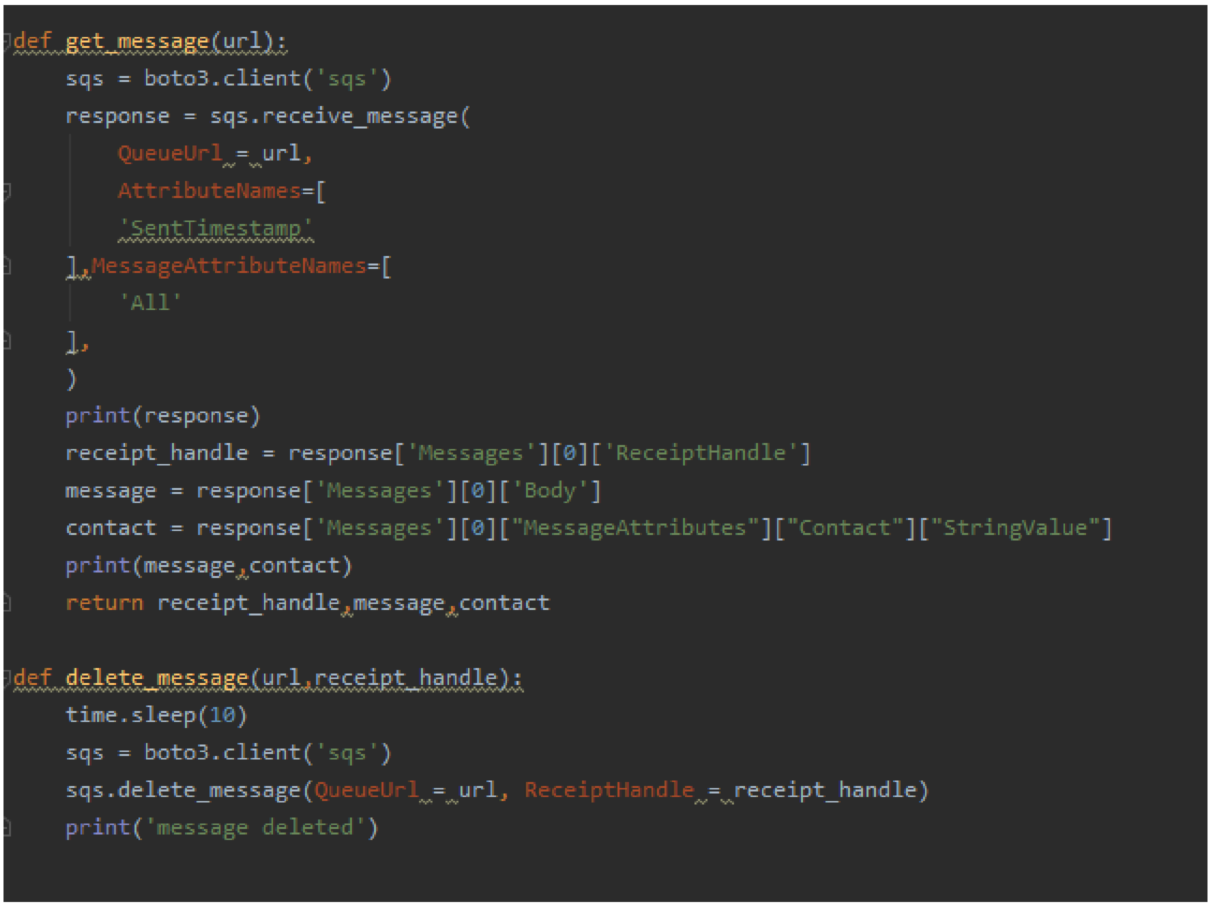
The above image shows the creation of random orders containing order items and their quantity. A random_message() function is called which returns a message.



```python
def get_queue_url(queue_name):
    sqs = boto3.client('sqs')
    response = sqs.get_queue_url(QueueName=queue_name)
    return response['QueueUrl']

def delete_queue(url):
    sqs = boto3.client('sqs')
    sqs.delete_queue(QueueUrl = url)


# 'https://sqs.us-east-1.amazonaws.com/832544204593/serverlessAssignment.fifo'
def send_message(url, messageAttribute, message):
    sqs = boto3.client('sqs')

    response = sqs.send_message(QueueUrl=url, DelaySeconds=10,MessageAttributes= messageAttribute,
    MessageBody=(
        message
        )
    )
    return response['MessageId']
```

Figure f send_message() [3]

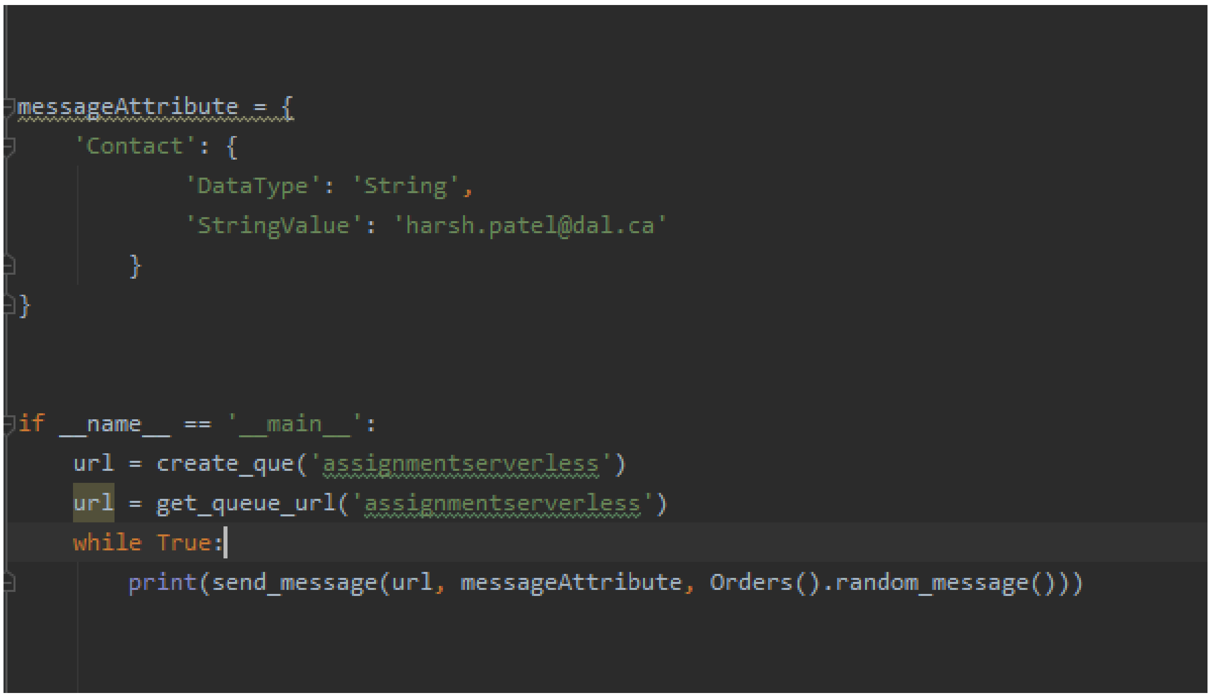
Above image contains 3 functions i.e. get_queue_url(), delete_queue() and send_message().



```
def get_message(url):
    sqs = boto3.client('sqs')
    response = sqs.receive_message(
        QueueUrl = url,
        AttributeNames=[
        'SentTimestamp'
    ],MessageAttributeNames=[
        'All'
    ],
    )
    print(response)
    receipt_handle = response['Messages'][0]['ReceiptHandle']
    message = response['Messages'][0]['Body']
    contact = response['Messages'][0]["MessageAttributes"]["Contact"]["StringValue"]
    print(message,contact)
    return receipt_handle,message,contact

def delete_message(url,receipt_handle):
    time.sleep(10)
    sqs = boto3.client('sqs')
    sqs.delete_message(QueueUrl = url, ReceiptHandle = receipt_handle)
    print('message deleted')
```

Figure g get_message() and delete_message() [3]

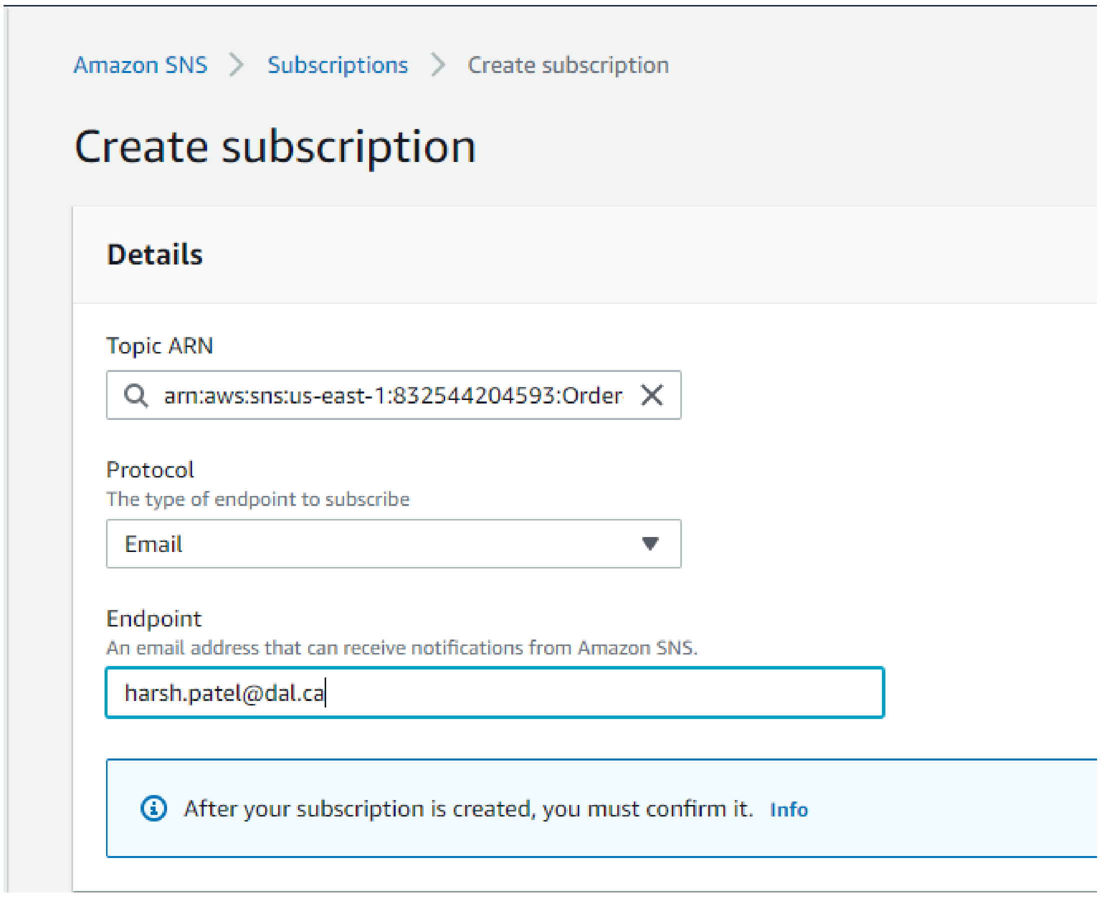
Above image contains functions for get_message() and delete_message().



```
messageAttribute = {
    'Contact': {
            'DataType': 'String',
            'StringValue': 'harsh.patel@dal.ca'
        }
}


if __name__ == '__main__':
    url = create_que('assignmentserverless')
    url = get_queue_url('assignmentserverless')
    while True:
        print(send_message(url, messageAttribute, Orders().random_message()))
```

Figure h Running SQS [3]

Finally, we will create a queue, get its URL, and keep on sending the messages. We can also keep a delay with time.sleep(300) to give a delay of 5 minutes. Thus, the Simple Queue Service part is complete.



Create subscription

Details

Topic ARN
arn:aws:sns:us-east-1:832544204593:Order

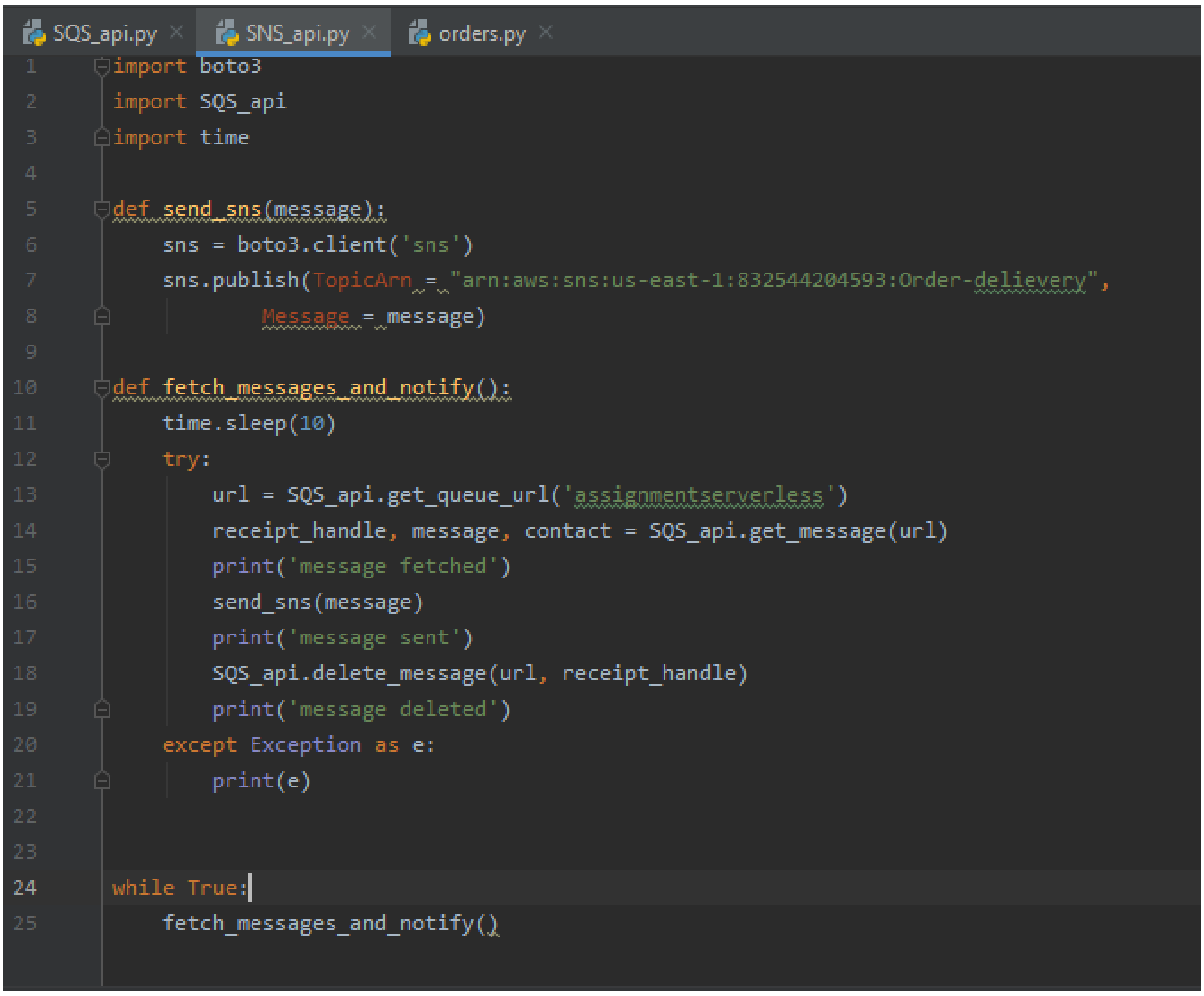
Protocol
The type of endpoint to subscribe
Email

Endpoint
An email address that can receive notifications from Amazon SNS.
harsh.patel@dal.ca

After your subscription is created, you must confirm it. Info

Figure i Creating a subscription [4]

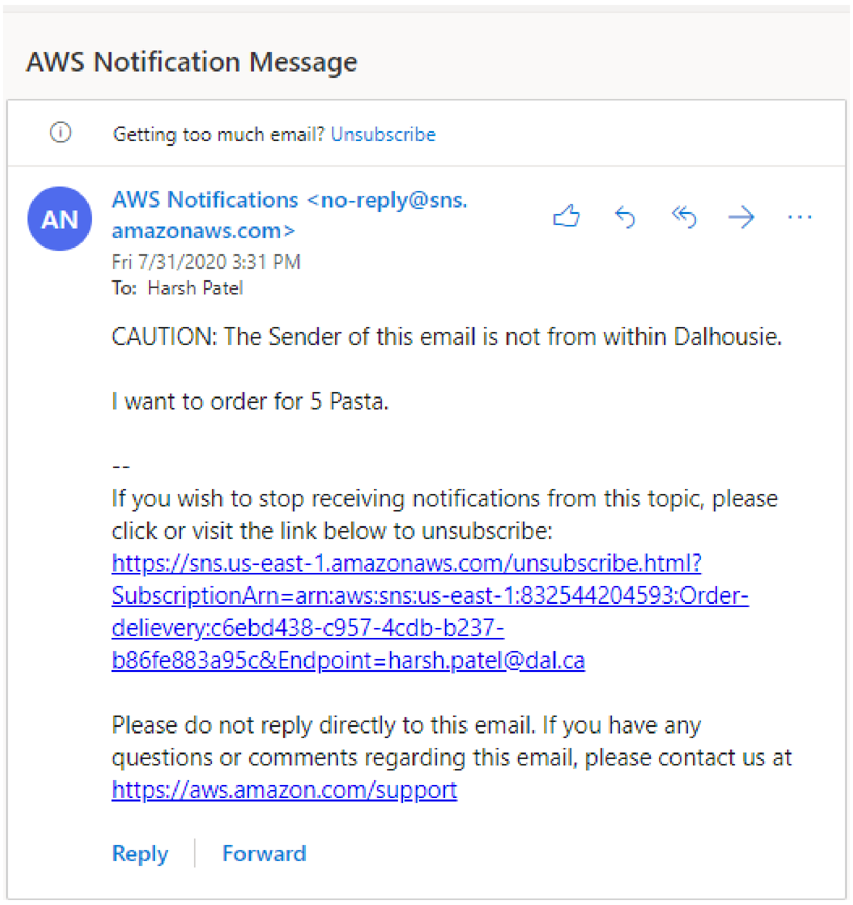
Now, I created a token for SNS and subscriptions for my email address.



```python
import boto3
import SQS_api
import time

def send_sns(message):
    sns = boto3.client('sns')
    sns.publish(TopicArn = "arn:aws:sns:us-east-1:832544204593:Order-delievery",
                Message = message)

def fetch_messages_and_notify():
    time.sleep(10)
    try:
        url = SQS_api.get_queue_url('assignmentserverless')
        receipt_handle, message, contact = SQS_api.get_message(url)
        print('message fetched')
        send_sns(message)
        print('message sent')
        SQS_api.delete_message(url, receipt_handle)
        print('message deleted')
    except Exception as e:
        print(e)


while True:
    fetch_messages_and_notify()
```

Figure j SNS API [4]

The above code fetches the messages every 10 seconds, deletes that message, and sends a notification to the subscriber.



AWS Notification Message

Getting too much email? Unsubscribe

AWS Notifications <no-reply@sns.amazonaws.com>
Fri 7/31/2020 3:31 PM
To: Harsh Patel

CAUTION: The Sender of this email is not from within Dalhousie.

I want to order for 5 Pasta.

--
If you wish to stop receiving notifications from this topic, please click or visit the link below to unsubscribe:
https://sns.us-east-1.amazonaws.com/unsubscribe.html?SubscriptionArn=arn:aws:sns:us-east-1:832544204593:Order-delievery:c6ebd438-c957-4cdb-b237-b86fe883a95c&Endpoint=harsh.patel@dal.ca

Please do not reply directly to this email. If you have any questions or comments regarding this email, please contact us at https://aws.amazon.com/support

Reply | Forward

Figure k SNS notify to the subscriber

References

[1] "Amazon Kinesis - Easily collect, process, and analyze video and data streams in real time," Amazon, [Online]. Available: https://aws.amazon.com/kinesis/. [Accessed 31 07 2020].
[2] "Weather API - Current & Forecast weather data collection," OpenWeather, [Online]. Available: https://openweathermap.org/api. [Accessed 31 07 2020].
[3] " Amazon Simple Queue Service," Amazon, [Online]. Available: https://aws.amazon.com/sqs/. [Accessed 31 07 2020].
[4] " Amazon Simple Notification Service - Fully managed pub/sub messaging for microservices, distributed systems, and serverless applications," Amazon, [Online]. Available: https://aws.amazon.com/sns/. [Accessed 31 07 2020].