Serverless versus traditional web hosting NLP Chatbot Analysis:

Sam Kececi and Justin Zwick

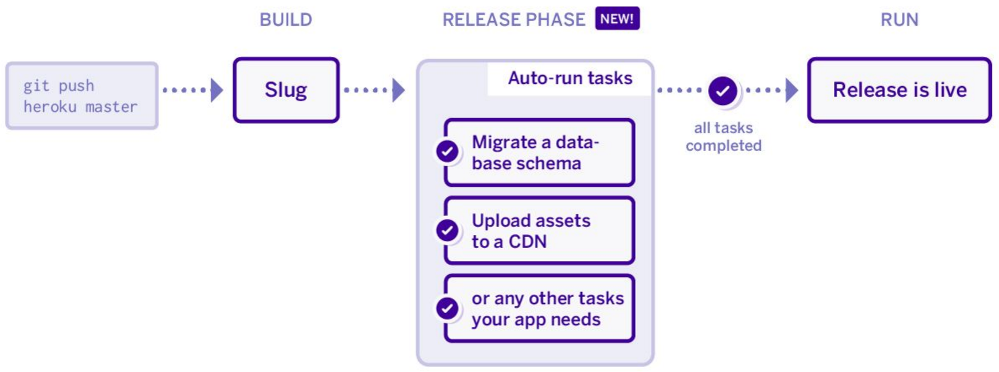
The main goal of our project is to examine the efficacy of building a chatbot using serverless technology. To do this, we chose to focus on AWS Lambda and build a chatbot using that platform. In order to examine the efficacy of using this technology for software engineering applications like these, we will compare chatbot built using the most popular serverless tool -- *AWS Lambda* -- with a more traditional method: a chatbot hosted on *Heroku*, one of the most popular cloud hosting platforms.

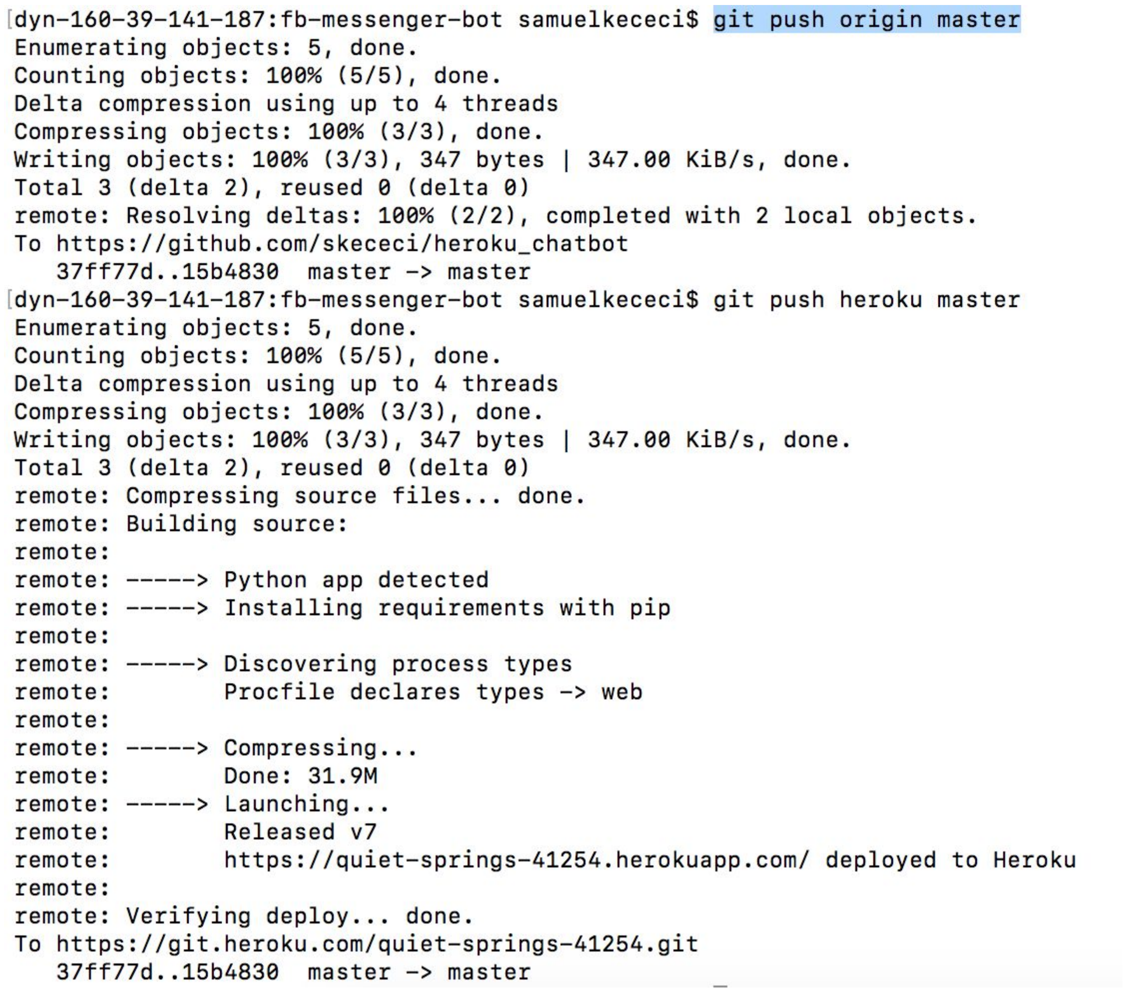
Up until now, there has never been a comprehensive study into the advantages and disadvantages of using serverless technology for a chatbot. Our study will provide insight into which system provides a better option to create a chatbot, serverless or a traditional server model. In other words, this project will analyze whether serverless technology is better suited for chatbots.

Intuitively, it would make sense that serverless would be perfect for chatbots due to the high amount of idle time of the application, the existence of a clearly defined ‘trigger’ (the sending of a message into Messenger), and a plethora of third-party vendors providing free NLP chat services to developers. This study will examine if reality actually reflects intuition.

This projects provides immense value to the software engineering community seeking to design chatbots. For nearly every engineer, there are several key metrics which must be researched when deciding on a technology. Chief among these are ease of development, cost effectiveness, and performance (in terms of both speed and storage space needed). Our project, focusing on AWS Lambda, will assess all of these key metrics and give results, but will be primarily focused on ease of development and the practicality of using these services. This will be extremely useful to any engineer seeking to decide between these hosting options.

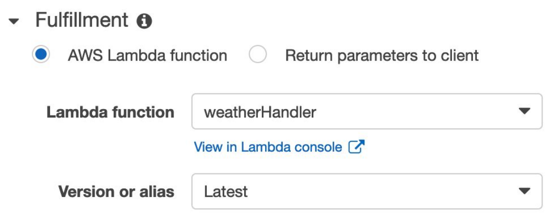
Outline of the Heroku development process:





```
[dyn-160-39-141-187:fb-messenger-bot samuelkececi$ git push origin master
 Enumerating objects: 5, done.
 Counting objects: 100% (5/5), done.
 Delta compression using up to 4 threads
 Compressing objects: 100% (3/3), done.
 Writing objects: 100% (3/3), 347 bytes | 347.00 KiB/s, done.
 Total 3 (delta 2), reused 0 (delta 0)
 remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
 To https://github.com/skececi/heroku_chatbot
    37ff77d..15b4830  master -> master
[dyn-160-39-141-187:fb-messenger-bot samuelkececi$ git push heroku master
 Enumerating objects: 5, done.
 Counting objects: 100% (5/5), done.
 Delta compression using up to 4 threads
 Compressing objects: 100% (3/3), done.
 Writing objects: 100% (3/3), 347 bytes | 347.00 KiB/s, done.
 Total 3 (delta 2), reused 0 (delta 0)
 remote: Compressing source files... done.
 remote: Building source:
 remote:
 remote: -----> Python app detected
 remote: -----> Installing requirements with pip
 remote:
 remote: -----> Discovering process types
 remote:        Procfile declares types -> web
 remote:
 remote: -----> Compressing...
 remote:        Done: 31.9M
 remote: -----> Launching...
 remote:        Released v7
 remote:        https://quiet-springs-41254.herokuapp.com/ deployed to Heroku
 remote:
 remote: Verifying deploy... done.
 To https://git.heroku.com/quiet-springs-41254.git
    37ff77d..15b4830  master -> master
```

- Development process overview when pushing to Heroku:
    - Code for chatbot is committed to github
    - Then code is pushed to Heroku
    - On Heroku, all the dependencies are calculated, then they are installed as necessary (in this case pip is used)
    - Finally, the code is compiled and run on the server
    - In this case, the code is run continuously on the server, which in turn connects to Lex (which receives messages from facebook messenger through a webhook). So whenever a message is sent to the chatbot, it then forwards that message to the Lex service, which in turn sends a JSON message via HTTP with keywords to the the heroku server and a response is sent back the same way.
- Development process overview with Lambda:
    - Code for chatbot is written directly inside the browser in Lambda's code console
    - You can not simply import libraries willy-nilly when using Lambda, and since it isn't a traditional server you can't simply "pip install" them. When setting this up, I figured out that we must actually download the code for a package into a local directory on your machine, zip that, and then upload on the Lambda console in order to use that package. We will go over various other quirks in the final report when talking about the Lambda development process.
    - Lambda offers native integration with Lex, so whenever Lex receives a message it automatically forwards it to the Lambda function that is associated with that Lex *intent*
    - Whenever a message is sent to the chatbot, it then forwards that message to the Lex service via a webhook, which in turn sends a JSON message with keywords to the the heroku server and a response is sent back.
    - There are many Lambda functions, each associated with a different *intent* using the AWS console.



Fulfillment

[✓ AWS Lambda function] Return parameters to client

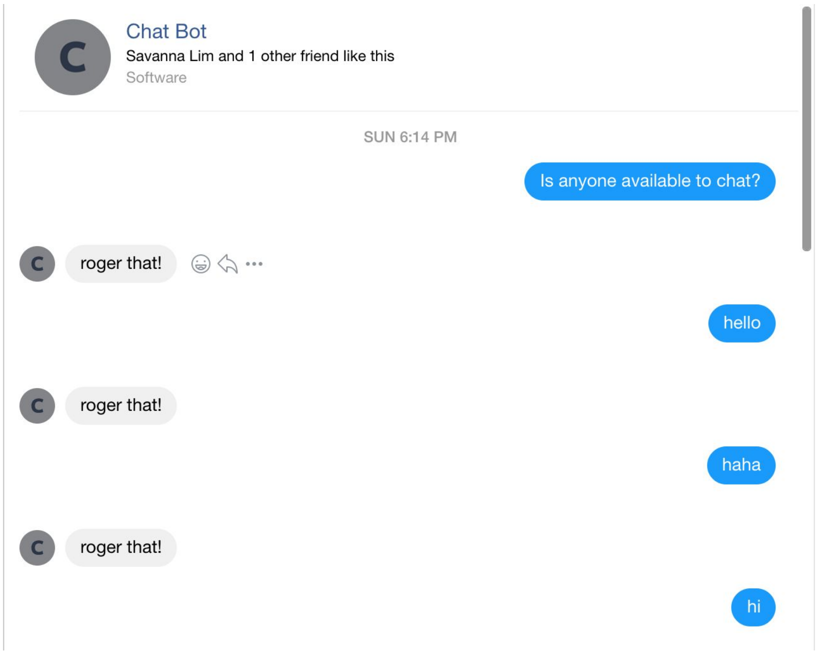
Lambda function: weatherHandler

View in Lambda console

Version or alias: Latest

Currently, we have (admittedly very simple) working versions of both chatbots that allows them to respond to messages sent to them with preset responses. Additionally, both chatbots are up and running and are hosted on AWS Lambda and Heroku respectively.

Chat bot hosted on heroku: *currently responds to all messages with "roger that!"*



Chat Bot
Savanna Lim and 1 other friend like this
Software

SUN 6:14 PM

Is anyone available to chat?

roger that!

hello

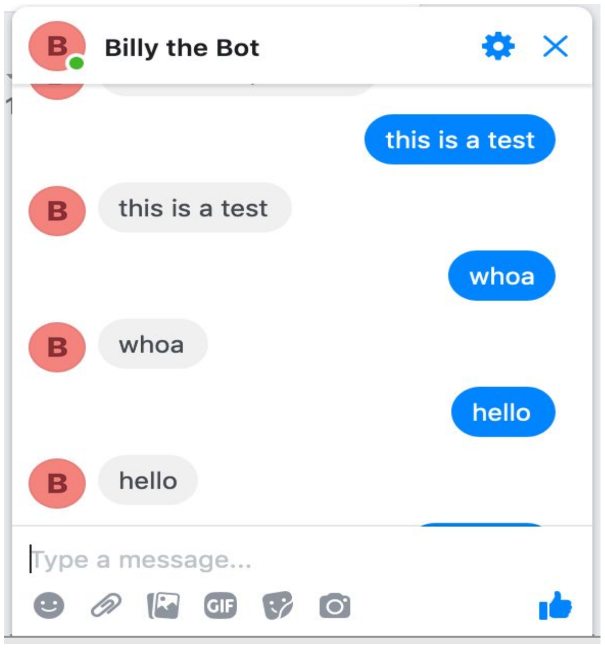
roger that!

haha

roger that!

hi

Chat bot built using AWS Lambda: *currently just echos back whatever message you send to it*



Billy the Bot

this is a test

this is a test

whoa

whoa

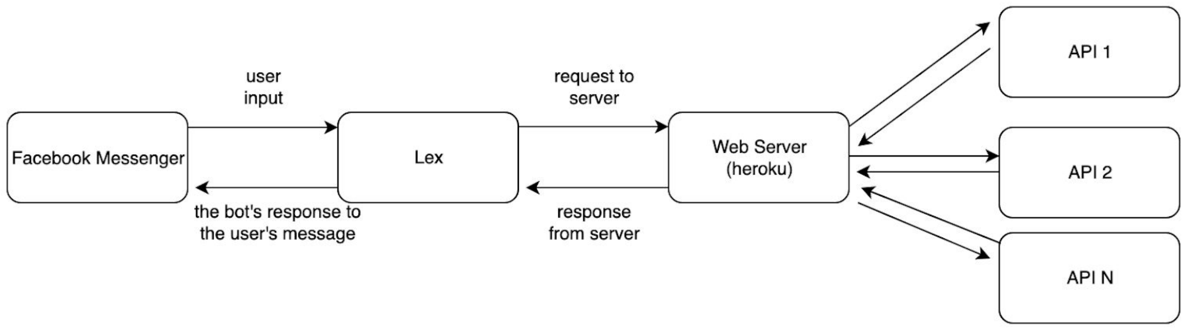
hello

hello

Type a message...

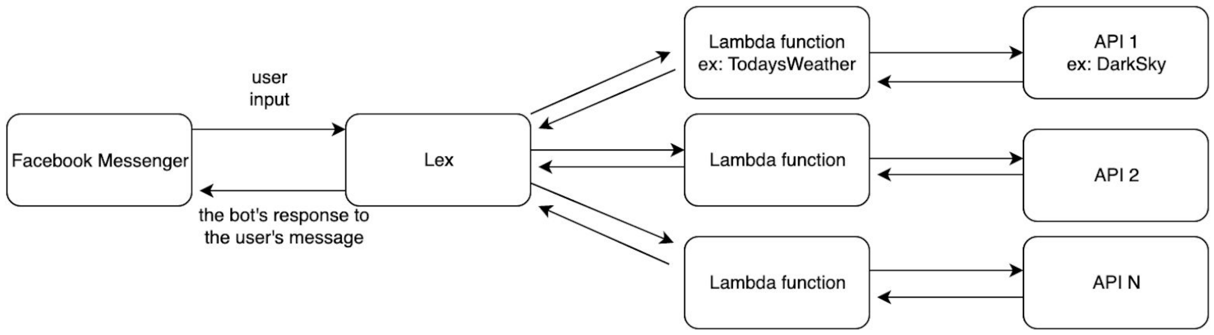
Although we haven't implemented the NLP conversational aspects of the chatbot with Lex yet, we do have the basic architecture and infrastructure already set up and working.

The architecture of the **monolithic server** bot is as follows:



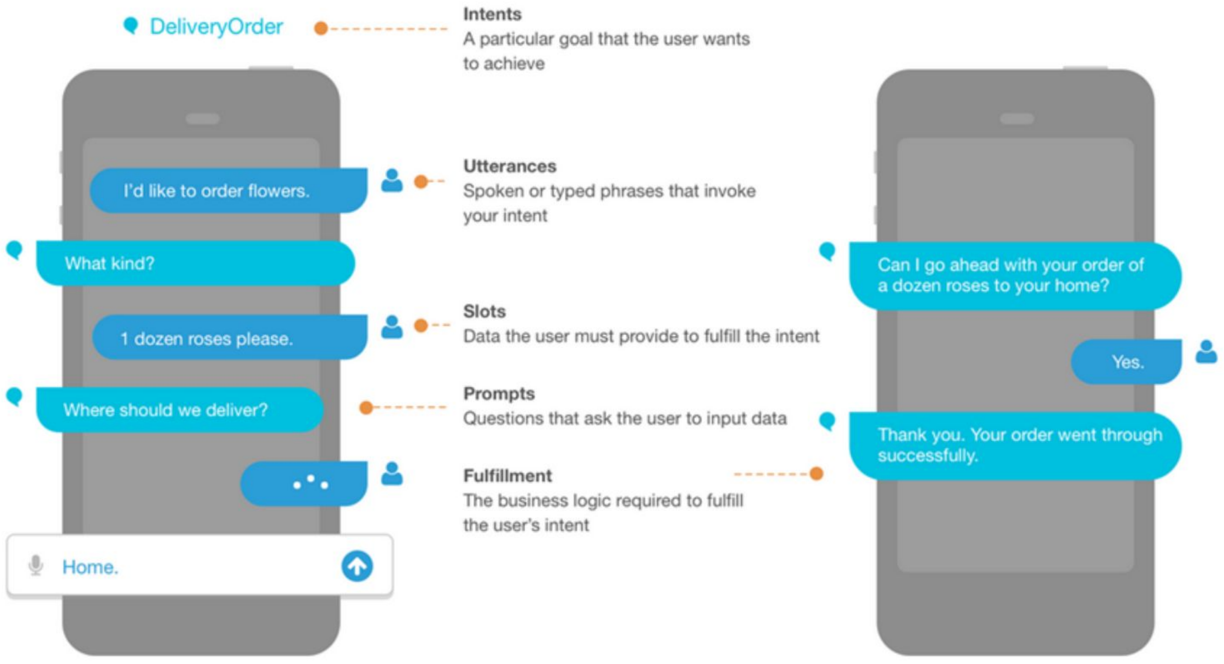
We chose to use Lex to integrate Natural Language Understanding and conversational fluency into our chatbots. When a message is sent from the user into FB Messenger, that same message is sent to Lex via a webhook. It is then classified into a series of different possible *intents,* which for our purpose are simply different possible conversational topics (the weather, traffic, etc.) which all correspond to different APIs we will be using.
The web server must orchestrate interactions with many different APIs directly, and will check the intent name that is encoded in the JSON message when deciding what it needs to do.

The architecture of the **serverless** bot is as follows:

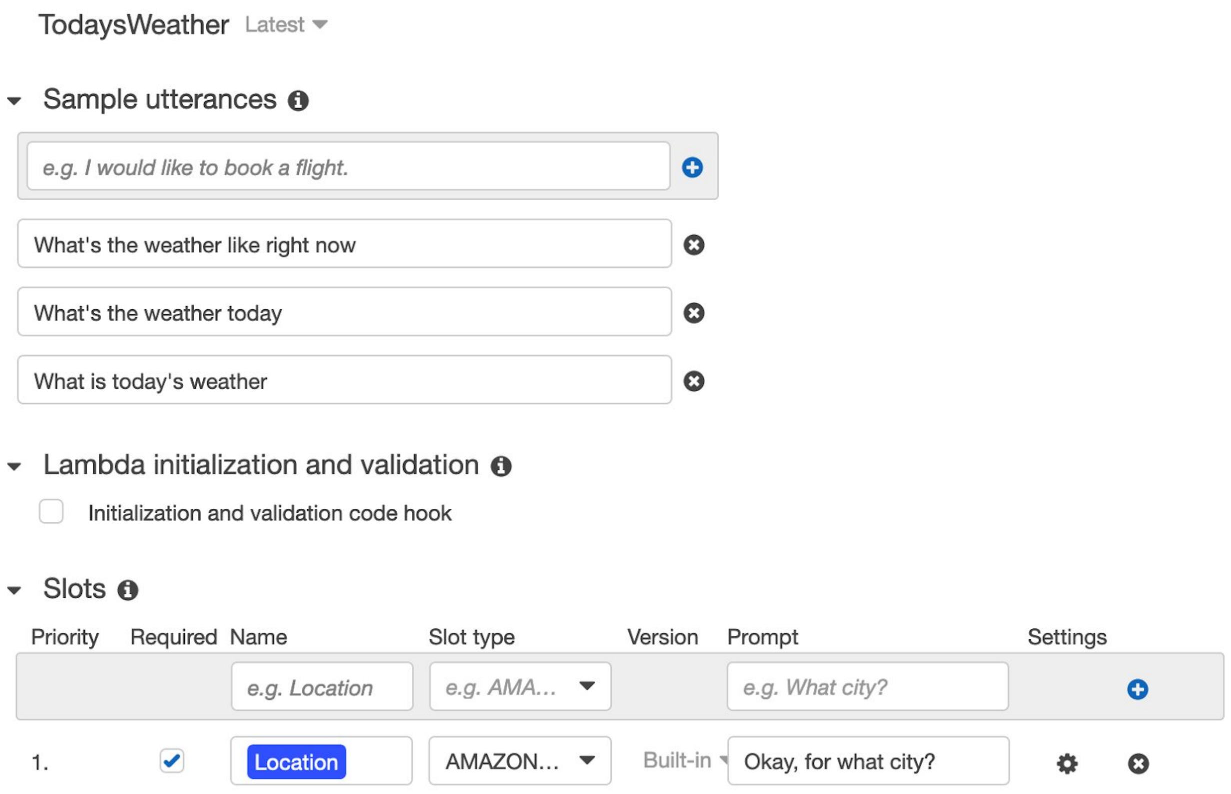


In this case, each of the Lex Intents are associated with a different Lambda function, which each correspond to a different topic of conversation. One thing we will probably analyze in our final report is the difficulty of integrating additional intents in serverless architecture.
We have mentioned intents multiple times but have not yet defined them. According to the official documentation, they are "an intent represents a goal that the bot's user wants to achieve (buying a plane ticket, scheduling an appointment, or getting a weather forecast, and so forth)."

The workflow of Lex is as follows: (picture taken from the official Lex documentation)


DeliveryOrder
Intents
A particular goal that the user wants to achieve
I'd like to order flowers.
Utterances
Spoken or typed phrases that invoke your intent
What kind?
1 dozen roses please.
Slots
Data the user must provide to fulfill the intent
Where should we deliver?
Prompts
Questions that ask the user to input data
Fulfillment
The business logic required to fulfill the user's intent
Home.
Can I go ahead with your order of a dozen roses to your home?
Yes.
Thank you. Your order went through successfully.

Here is the corresponding console for our TodaysWeather intent:


TodaysWeather Latest

Sample utterances

e.g. I would like to book a flight.

What's the weather like right now

What's the weather today

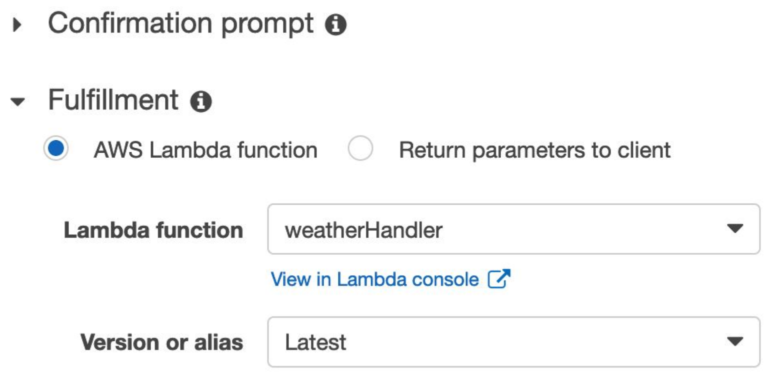
What is today's weather

Lambda initialization and validation

Initialization and validation code hook

Slots

Priority	Required	Name	Slot type	Version	Prompt	Settings
		e.g. Location	e.g. AMA...		e.g. What city?	
1.	✓	Location	AMAZON...	Built-in	Okay, for what city?	


▸ Confirmation prompt

▾ Fulfillment

[✓ AWS Lambda function] Return parameters to client

Lambda function: weatherHandler

View in Lambda console

Version or alias: Latest

We plan on posting all of our code publicly on github, and will make the repositories open source so that anyone can have insight into our results and also improve upon our chatbots.

The division of work is as follows:
Justin is responsible for creating the chatbot hosted on Heroku, and Sam is responsible for the AWS Lambda chatbot. Sam has no background in AWS Lambda and Justin has no background in Heroku, so it will be a learning process for both and both will be approached from a completely fresh perspective. By separately creating the chatbots, it will allow us to accurately assess the ease of development from a organic perspective.
We will both work together to develop metrics and compile the reports.