AWS Chatbot Challenge

Amazon Lex and AWS Lambda

Benjamin Towner

Cloud Solutions Architect / Engineer

[AWS Chatbot Challenge 2](#_Toc488137430)

[ChatBot Design Concept 3](#_Toc488137431)

[Overview 3](#_Toc488137432)

[Uniqueness 3](#_Toc488137433)

[Design Components 3](#_Toc488137434)

[Contact Management 3](#_Toc488137435)

[Activity Approval 4](#_Toc488137436)

[Calendar Events 4](#_Toc488137437)

[Models 4](#_Toc488137438)

[Intents 4](#_Toc488137439)

[Lambda 5](#_Toc488137440)

[MotherBot Blueprint 6](#_Toc488137441)

[Channels 7](#_Toc488137442)

[Facebook 7](#_Toc488137443)

[Slack 7](#_Toc488137444)

[Twillio 7](#_Toc488137445)

[Costs 7](#_Toc488137446)

[Entry Requirements 7](#_Toc488137447)

[Demo Video 7](#_Toc488137448)

[Github 7](#_Toc488137449)

[Access to Working Bot 7](#_Toc488137450)

[Testing Instructions 7](#_Toc488137451)

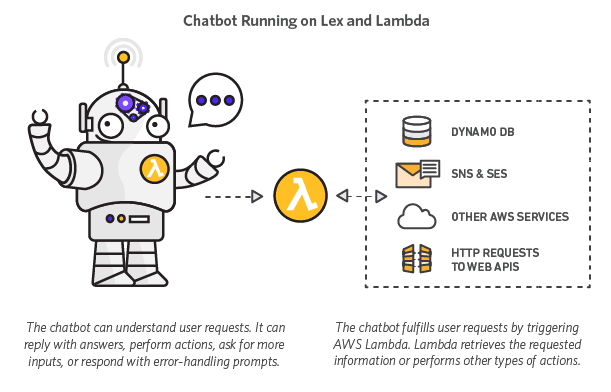
[Text Description 7](#_Toc488137452)

[Devpost Submission 7](#_Toc488137453)

# AWS Chatbot Challenge

Chatbots are changing how companies interface with their customers. With chatbots, you can easily fulfill the needs of your customers in an automated way using natural, human-like chat interfaces. Chatbots serve a variety of use cases, such as customer support, transaction fulfillment, data retrieval, or even DevOps functions (ChatOps).

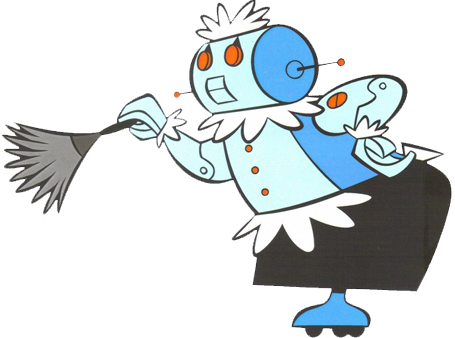
However, building and running chatbots is a difficult task. First, most developers lack the deep learning expertise necessary to create bots that can intelligently interpret and respond to text. In addition, developers must also provision, manage, and scale the compute resources necessary to run the bot’s code.



What if you could build chatbots with sophisticated natural language processing and almost no operational overhead? [Amazon Lex](https://aws.amazon.com/lex/) is a fully managed service for building conversational interfaces into any application using voice and text. Lex is powered by the same deep learning technologies that power Amazon Alexa and lets you build natural language chatbots. Lex is integrated with [AWS Lambda](https://aws.amazon.com/lambda/), a service that lets you run code without provisioning or managing servers. Lambda enables you to write and run logic for your chatbot using serverless compute. Getting started with Amazon Lex and AWS Lambda is quick and easy.

# ChatBot Design Concept

## Overview

**MotherBot** helps by Sharing Calendars, Approved Contacts and provides a self-service administration platform for the *C.E.O. of the Home* to effectively schedule and organize. The AWS Chat Bot allows various processes to be made available to these household with the primary purpose of verifying the identity of the individual connections, formalizing an approval process and managing the ‘feasibility’ on the calendar.

## Uniqueness

Households are like mini-organizations, except their members or ‘***little users***’ often find themselves in a place of constant challenge and response. Seeking approval and then finding out how to facilitate getting it done is the world that ‘***Tweens’*** live in. The coordination required among Parents often requires effective communication practices regardless of your Work-life or Marital status. Technology can enable Parents or Parental guardians to manage the household workloads while providing an appropriate level of privacy and respect.

# Design Components

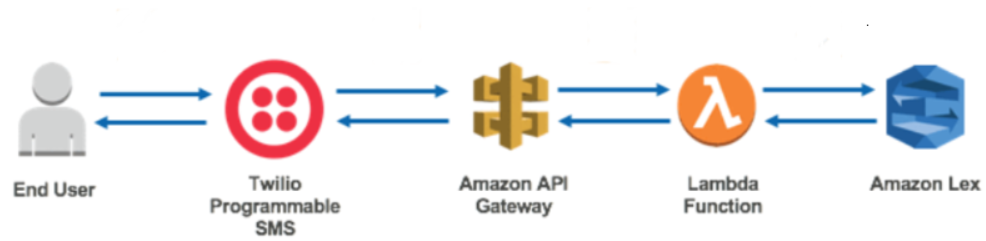
Reference a traditional 3 tiered Application stack (Web, Application and Data), various components are refactored to assume the functional responsibility.

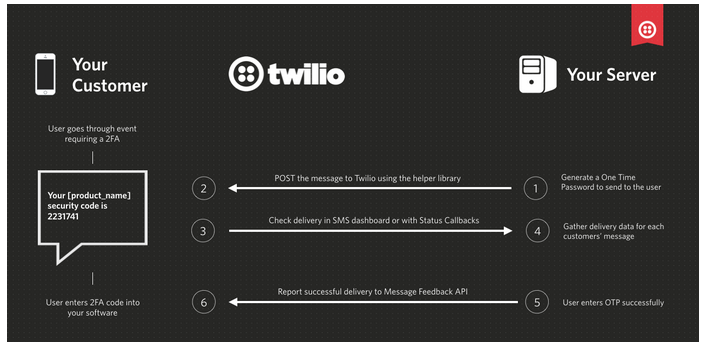
There are 3 main functional aspects to **MotherBot**.

1. Identity and Access Management (Contact Management)
2. Activity Approval
3. Calendar Feasibility

## Contact Management

**Phone Verification**, using Twillio, is used to make sure people are who they say they are when signing up as a ‘Friend’. It reduces risk and prevents fraudulent signups and increases trust. Phone verification events are best implemented when the user is giving you their number for the first time. When a user triggers that event, [generate a One Time Passcode (OTP)](https://en.wikipedia.org/wiki/One-time_password#Methods_of_generating_the_OTP) and send it via [SMS to the user](https://www.twilio.com/docs/api/rest/sending-messages) Present the user with a UI to enter the code and verify the code against the one you generated to see if they match.





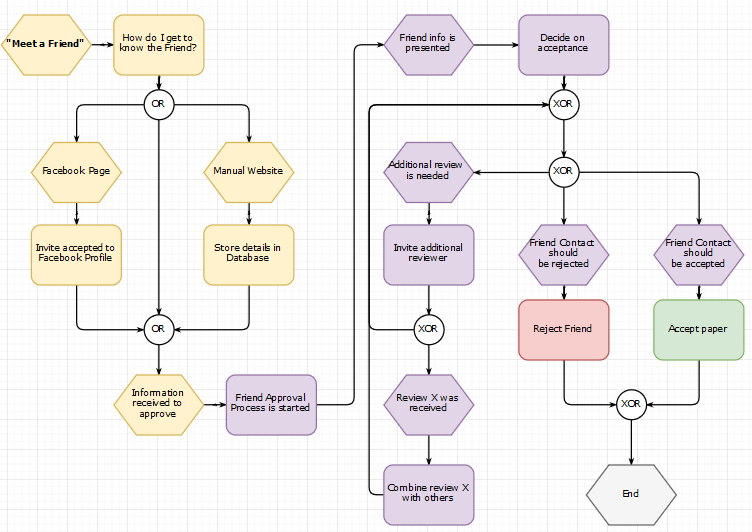
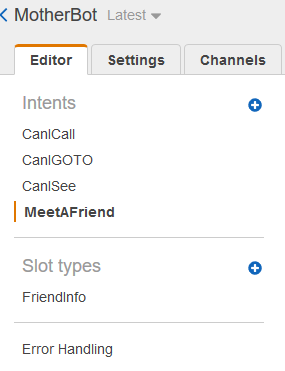
## Activity Approval

## Calendar Events

# Models

## Intents

* MeetAFriend
  + **Slots:** FriendInfo
  + **Utterances:**
  + **Prompts:**
* CanICall
  + **Slots:** WhoCall
  + **Utterances:**
  + **Prompts:**
* CanIGoTo:
  + **Slots:** ApprovedFriendsHome - Available approved Friend's House
  + **Slots:** ApprovedPublicPlaces – Pool Club, Library, Bowling, Mall
  + **Utterances:**
  + **Prompts:**
* CanISee:
  + **Slots:** Events –
  + **Slots:** Approved Movie
  + **Slots:** Approved Concert
  + **Utterances:**
  + **Prompts:**



## Lambda

Re-using some examples from other blueprints, the attached ’lex-motherbot-python.py’ can be used to create a Lambda function. This function uses the lambda\_handler to route functions based on the intent.

* Blueprints
  + lex-make-appointment
* Response Cards
* Lambda Functions as Code Hooks
* Lex- motherbot -python

def lambda\_handler(event, context):

"""

Route the incoming request based on intent.

The JSON body of the request is provided in the event slot.

"""

# By default, treat the user request as coming from the America/New\_York time zone.

os.environ['TZ'] = 'America/New\_York'

time.tzset()

logger.debug('event.bot.name={}'.format(event['bot']['name']))

return dispatch(event)

|  |  |  |
| --- | --- | --- |
| * + **elicit\_slot**   + **confirm\_intent**   + **close**   + **delegate**   + **build\_response\_card**   + **parse\_int**   + **try\_ex** | * + **increment\_time\_by\_thirty\_mins**   + **get\_random\_int**   + **get\_availabilities**   + **isvalid\_date**   + **is\_available**   + **get\_duration**   + **get\_availabilities\_for\_duration** | * + **build\_validation\_result**   + **validate\_book\_appointment**   + **build\_time\_output\_string**   + **build\_available\_time\_string**   + **build\_options**   + **make\_appointment**   + **dispatch**   + **lambda\_handler** |

 Test the Lambda function. You invoke the Lambda function twice, using sample data for both booking a car and booking a hotel.

1. Choose **Actions**, **Configure test event**.
2. Choose **Lex-Book Hotel (preview)** from the **Sample event template** list.

This sample event matches the Amazon Lex request/response model. For more information, see [Using Lambda Functions](http://docs.aws.amazon.com/lex/latest/dg/using-lambda.html).

1. Choose **Save and test**.
2. Verify that the Lambda function successfully executed. The response in this case matches the Amazon Lex response model.
3. Repeat the step. This time you choose the **Lex-Book Car (preview)** from the **Sample event template** list. The Lambda function processes the car reservation.

# MotherBot Blueprint

* **Intent** – (MeetAFriend), which is preconfigured as follows:
  + **Slots** – The intent is configured with the following slots:
    - Slot AppointmentType, of the AppointmentTypes custom type.
    - Slot Date, of the AMAZON.DATE built-in type.
    - Slot Time, of the AMAZON.TIME built-in type.
  + **Utterances** – The intent is preconfigured with the following utterances:
    - "I would like to book an appointment"
    - "Book an appointment"
    - "Book a {AppointmentType}"

If the user utters any of these, Amazon Lex determines that MakeAppointment is the intent, and then uses the prompts to elicit slot data.

* + **Prompts** – The intent is preconfigured with the following prompts:
    - Prompt for the AppointmentType slot – "What type of appointment would you like to schedule?"
    - Prompt for the Date slot – "When should I schedule your {AppointmentType}?"
    - Prompt for the Time slot – "At what time do you want to schedule the {AppointmentType}?" and

"At what time on {Date}?"

* + - Confirmation prompt – "{Time} is available, should I go ahead and book your appointment?"
    - Cancel message– "Okay, I will not schedule an appointment."

# Channels

Facebook**-**

Slack**-**

Twillio**-**

* Appointment Reminders
* 2-Factor Authentication for Friend Contacts
* Phone number verification
* Arrival Alerts

# Costs

# Entry Requirements

1. A **demo video** that clearly shows your bot functioning on its intended platform.
2. **Share your code** repository publicly or privately through GitHub or BitBucket with testing@devpost.com.
3. **Access to a working bot**.
4. **Testing instructions** with anything we need to know to test your bot.
5. **Text description** - include a brief explanation of what the bot Application does and what makes it unique.
6. A **completed submission form** on Devpost.

## Demo Video

Created using Skype for business, and produce the publish location.

## Github

https://github.com/bentowner/MotherBot

## Access to Working Bot

## Testing Instructions

Follow the workflow from this Document

## Text Description

This document will serve as the Text Description of MotherBot, the Challenge and the Submission.

## Devpost Submission