

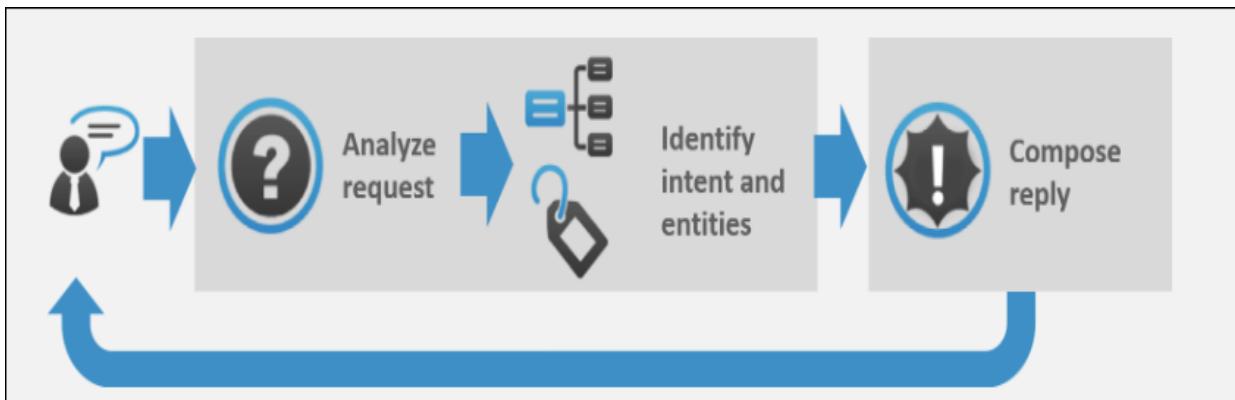
Building A ChatBot With Amazon Lex

Chatbot Concept

What are chatbots?

A Chatbots is an intelligent software that can simulate a conversation (or a chat) with a user in natural language through messaging applications, websites, mobile phones or telephone. They can be programmed to respond to simple keywords or prompts to complex conversations.

A Chatbot has two different tasks at the core: user request analysis (identifying the intent of the user) and returning the response.



There are two types of chatbots: one functions on a set of rules and the other more advanced one uses artificial intelligence.

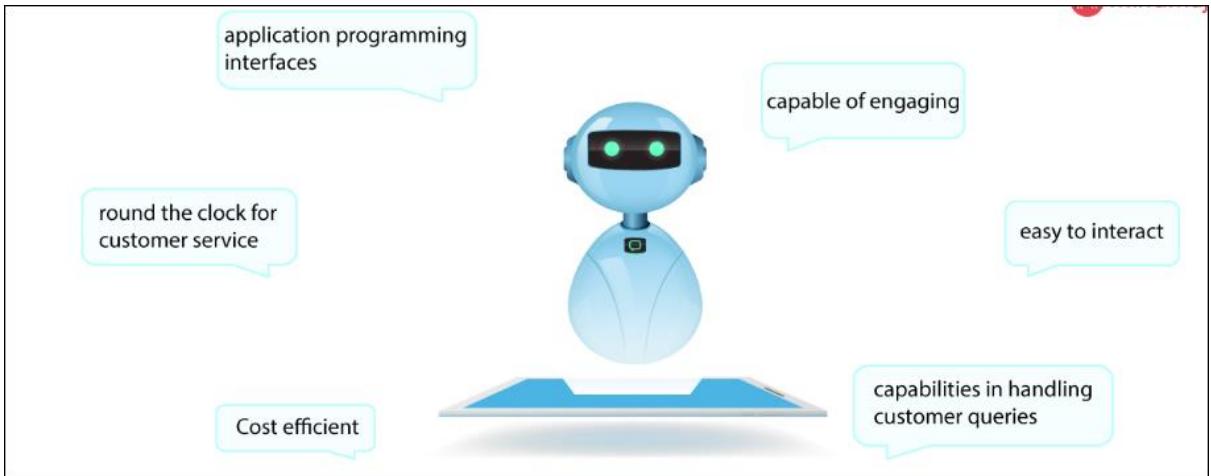
1. **Chatbots that function on rules:** It is easy to build. Though simplistic, it can get basics tasks done with limitations to specific commands. This bot is only as smart as it is programmed.
2. **Chatbots that function on machine learning:** It understands languages and not just commands and provides relevant responses. It communicates through speech or text relying on AI technologies like machine learning and NLP. It gets smarter as it learns from the conversations from users.

Use cases

A chatbot can be used in many different ways. Any business or industry having a website or an app can have a chatbot. Following are some of the chatbot applications, out of the infinite possibilities:

1. A takeaway restaurant that allows customers to order from their office or home.
2. A chatbot that answers customer service questions and provides help with different tasks.
3. A chatbot that allows the customer to book a flight and receive relevant information.
4. A chatbot that helps a customer with ecommerce purchases.
5. A marketing campaign chatbot that asks questions to the customer.
6. A health bot that provides services for booking a Doctor consultation.
7. A movie ticket booking bot.

Advantages of Chatbots



The advantages of chatbots are as follows:

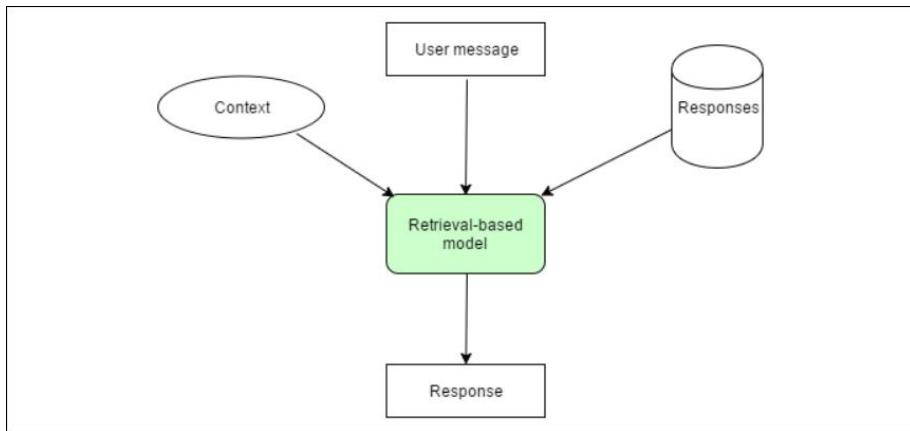
- They are capable of engaging customers in a friendly manner.
- A simple interface makes it easier to interact with.
- Interactions are made possible via social media platforms like Twitter, Facebook, etc. Chats are served through application programming interfaces (APIs).
- Chatbots have extraordinary capabilities in handling customer queries.
- Chatbots work efficiently round the clock for customer service.
- Cost-efficient and are easy to build.

Models for a Chatbot

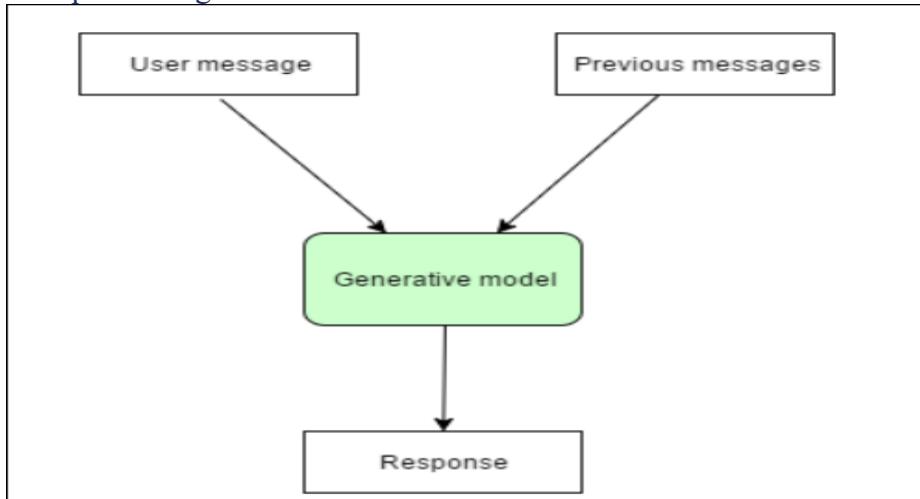
When creating a chatbot, the goal is to automate the process altogether and to reduce human intervention. The first step in creating a chatbot involves inputting thousands of existing interactions between customers and customer service representatives to teach the machine which words/phrases are sensitive to the business. The next step is preprocessing which involves incorporating grammar to identify any misspellings.

Then the next step is to identify the kind of chatbot we would like to implement. There mainly are two models:

1. **Retrieval-based model**- These models are much easy to build and give more predictable responses. They make use of context in the conversation for selecting the best response from a predefined list of messages that they got trained from. The context includes all the previous conversations, and the saved variables.



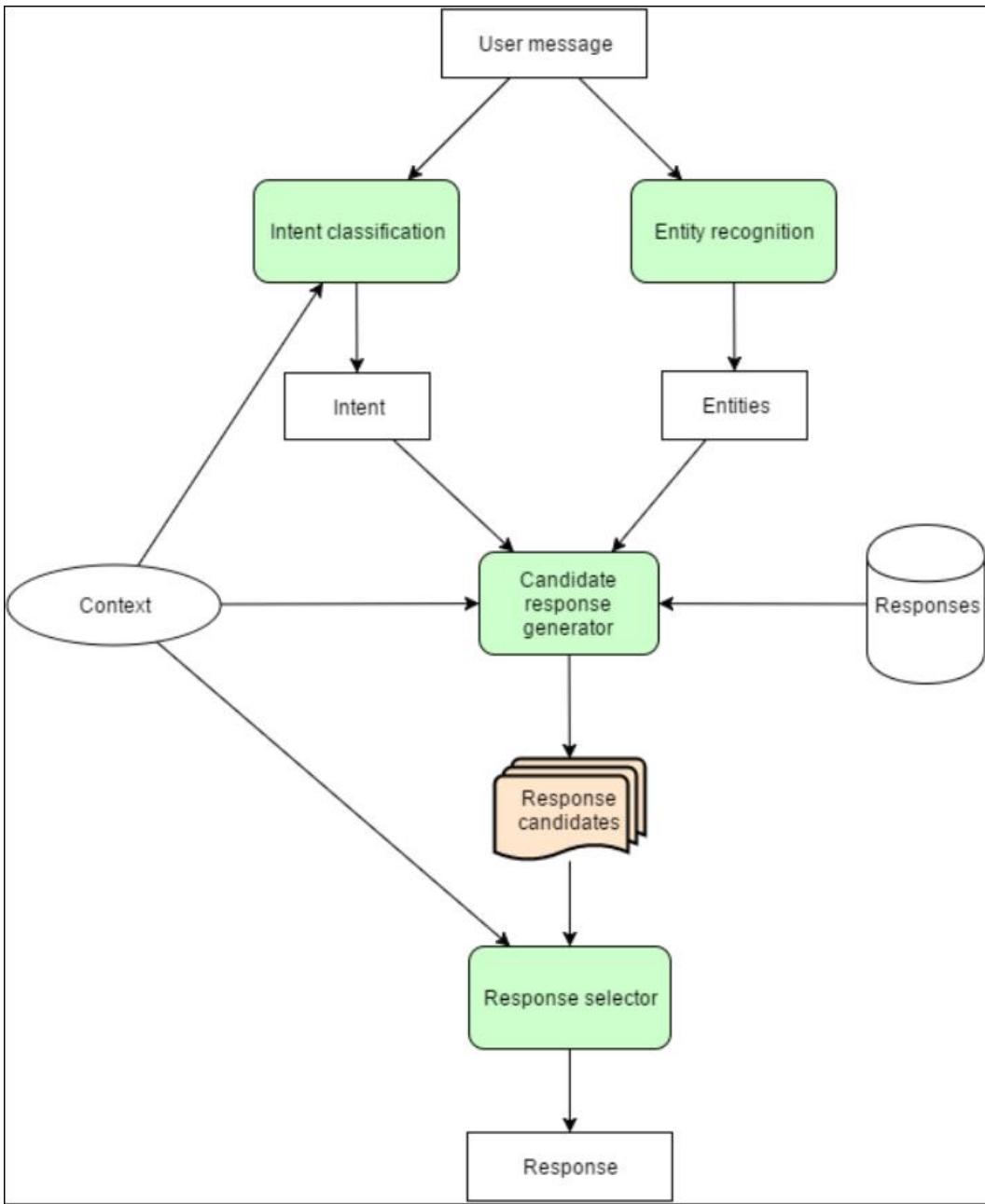
2. **Generative based model:** A generative model chatbot doesn't use any predefined repository. This kind of chatbot is more advanced because it learns from scratch using “Deep Learning.”



Pros and Cons:

Generative models	Selective models
<ul style="list-style-type: none"> + Can possibly generate arbitrary answer (more similar to general AI) + Can generate answer in correct grammar form (e.g. with correct speaker gender) - Can generate answer with incorrect grammar/syntax - Prone to “general answer” problem - Difficult to impose properties on model replies (e.g. no obscene words, speak like some specific person), but possible! 	<ul style="list-style-type: none"> - Restricted pool of answers which can not cover all dialogue topics - For context “What is your name, girl?” can select “My name is Stephen.” (inconsistency) + Predefined answers have good grammar/syntax + Less prone to “general answer” problems + You can customize answers for your own needs (without obscenities, kind answers)

Architecture of AI chatbot



Message processing begins with understanding what the user is talking about.

1. An intent classification module identifies the intent of the user messages. Typically, it is a selection of one out of many predefined intents, though more sophisticated bots can identify multiple intents from one message.
2. An entity recognition module extracts structured bits of information from the message. The weather bot can extract the location and date.

3. The candidate response generator is doing all the domain-specific calculations to process the user request. It can use different algorithms, call a few external APIs, or even ask a human to help with response generation. The result of these calculations is a list of response candidates. All these responses should be correct according to the domain-specific logic. They can't be just tons of random responses. The response generator must use the context of the conversation as well as intent and entities extracted from the last user message. Otherwise, it can't support multi-message conversations.

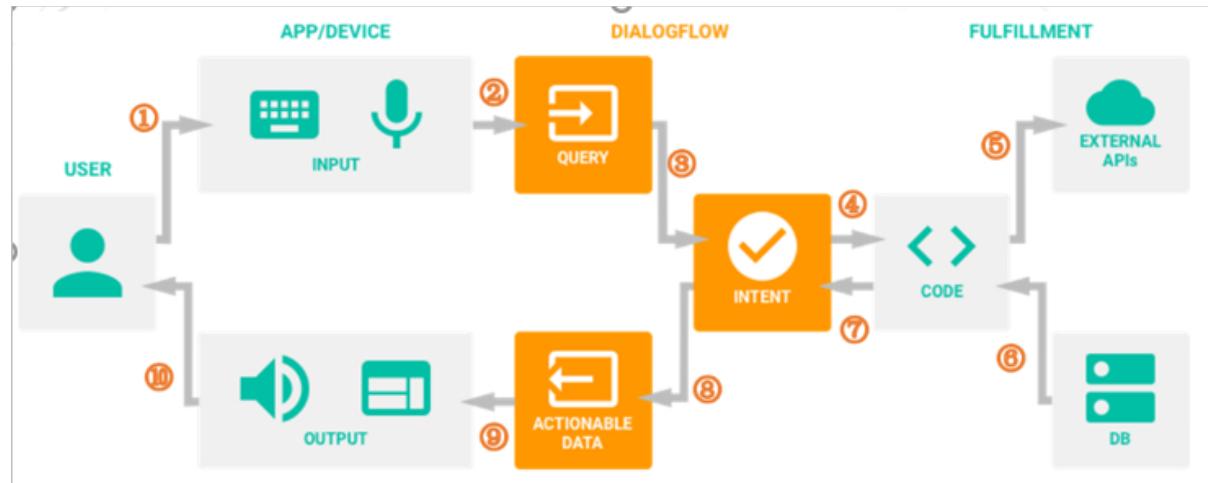
4. The response selector just scores all the response candidate and selects a response which should work better for the user.

Amazon Lex

Introduction

Amazon Lex is an AWS service for building conversational interfaces for applications using voice and text.

How do Chatbots work?



Detailed steps:

1. A user sends a text/voice message to a device or an App
2. The App/Device transfers the message to Lex
3. The message is categorized and matched to a corresponding intent (Intents are defined manually by developers in Lex)
4. We define actions for each intent in the corresponding fulfillment (Webhook).
5. When Lex finds a specific intent, the webhook will use external APIs to find a response from external databases.
6. The external databases send back the required information to the webhook.
7. Webhook sends a formatted response to the intent.

8. Intent generates actionable data according to different channels.
9. The actionable data go to output Apps/Devices.
10. The user gets a text/image/voice response.

What Is Amazon Lex?

Amazon Lex is an AWS service for building conversational interfaces for applications using voice and text. With Amazon Lex, the same conversational engine that powers Amazon Alexa is now available to any developer, enabling you to build sophisticated, natural language chatbots into your new and existing applications. Amazon Lex provides the deep functionality and flexibility of natural language understanding (NLU) and automatic speech recognition (ASR) so you can build highly engaging user experiences with lifelike, conversational interactions, and create new categories of products.

Amazon Lex enables any developer to build conversational chatbots quickly. With Amazon Lex, no deep learning expertise is necessary—to create a bot, you just specify the basic conversation flow in the Amazon Lex console. Amazon Lex manages the dialogue and dynamically adjusts the responses in the conversation. Using the console, you can build, test, and publish your text or voice chatbot. You can then add the conversational interfaces to bots on mobile devices, web applications, and chat platforms (for example, Facebook Messenger).

Amazon Lex provides pre-built integration with AWS Lambda, and you can easily integrate with many other services on the AWS platform, including Amazon Cognito, AWS Mobile Hub, Amazon CloudWatch, and Amazon DynamoDB. Integration with Lambda provides bots access to pre-built serverless enterprise connectors to link to data in SaaS applications, such as Salesforce, HubSpot, or Marketo.

Some of the benefits of using Amazon Lex include:

- **Simplicity** – Amazon Lex guides you through using the console to create your own chatbot in minutes. You supply just a few example phrases, and Amazon Lex builds a complete natural language model through which the bot can interact using voice and text to ask questions, get answers, and complete sophisticated tasks.
- **Democratized deep learning technologies** – Powered by the same technology as Alexa, Amazon Lex provides ASR and NLU technologies to create a Speech Language Understanding (SLU) system. Through SLU, Amazon Lex takes natural language speech and text input, understands the intent behind the input, and fulfills the user intent by invoking the appropriate business function.

Speech recognition and natural language understanding are some of the most challenging problems to solve in computer science, requiring sophisticated deep learning algorithms to be trained on massive amounts of data and infrastructure. Amazon Lex puts deep learning technologies within reach of all developers, powered by the same technology as Alexa. Amazon Lex chatbots convert incoming speech to text and understand the user intent to generate an intelligent response, so you can focus on building your bots with differentiated value-add for your customers, to define entirely new categories of products made possible through conversational interfaces.

- **Seamless deployment and scaling** – With Amazon Lex, you can build, test, and deploy your chatbots directly from the Amazon Lex console. Amazon Lex enables you to easily publish your voice or text chatbots for use on mobile devices, web apps, and chat services (for example, Facebook Messenger). Amazon Lex scales automatically so you don't need to worry about provisioning hardware and managing infrastructure to power your bot experience.
- **Built-in integration with the AWS platform** – Amazon Lex has native interoperability with other AWS services, such as Amazon Cognito, AWS Lambda, Amazon CloudWatch, and AWS Mobile Hub. You can take advantage of the power of the AWS platform for security, monitoring, user authentication, business logic, storage, and mobile app development.
- **Cost-effectiveness** – With Amazon Lex, there are no upfront costs or minimum fees. You are charged only for the text or speech requests that are made. The pay-as-you-go pricing and the low cost per request make the service a cost-effective way to build conversational interfaces. With the Amazon Lex free tier, you can easily try Amazon Lex without any initial investment.

Amazon Lex: How It Works

Amazon Lex enables you to build applications using a speech or text interface powered by the same technology that powers Amazon Alexa. Following are the typical steps you perform when working with Amazon Lex:

1. Create a bot and configure it with one or more intents that you want to support. Configure the bot so it understands the user's goal (intent), engages in conversation with the user to elicit information, and fulfills the user's intent.
2. Test the bot. You can use the test window client provided by the Amazon Lex console.

3. Publish a version and create an alias.
4. Deploy the bot. You can deploy the bot on platforms such as mobile applications or messaging platforms such as Facebook Messenger.

Before you get started, familiarize yourself with the following Amazon Lex core concepts and terminology:

- **Bot** – A bot performs automated tasks such as ordering a pizza, booking a hotel, ordering flowers, and so on. An Amazon Lex bot is powered by Automatic Speech Recognition (ASR) and Natural Language Understanding (NLU) capabilities, the same technology that powers Amazon Alexa.

Amazon Lex bots can understand user input provided with text or speech and converse in natural language. You can create Lambda functions and add them as code hooks in your intent configuration to perform user data validation and fulfillment tasks.

- **Intent** – An intent represents an action that the user wants to perform. You create a bot to support one or more related intents. For example, you might create a bot that orders pizza and drinks. For each intent, you provide the following required information:
 - Intent name – A descriptive name for the intent. For example, OrderPizza.
 - Sample utterances – How a user might convey the intent. For example, a user might say "Can I order a pizza please" or "I want to order a pizza".
 - How to fulfill the intent – How you want to fulfill the intent after the user provides the necessary information (for example, place order with a local pizza shop). We recommend that you create a Lambda function to fulfill the intent.

You can optionally configure the intent so Amazon Lex simply returns the information back to the client application to do the necessary fulfillment.

In addition to custom intents such as ordering a pizza, Amazon Lex also provides built-in intents to quickly set up your bot. For more information, see Built-in Intents and Slot Types.

- **Slot** – An intent can require zero or more slots or parameters. You add slots as part of the intent configuration. At runtime, Amazon Lex prompts the user for specific slot values. The user must provide values for all *required* slots before Amazon Lex can fulfill the intent.

For example, the OrderPizza intent requires slots such as pizza size, crust type, and number of pizzas. In the intent configuration, you add these slots. For each slot, you provide slot type and a prompt for Amazon Lex to send to the client to elicit data from the user. A user can reply with a slot value that includes additional words, such as "large pizza please" or "let's stick with small." Amazon Lex can still understand the intended slot value.

- **Slot type** – Each slot has a type. You can create your custom slot types or use built-in slot types. For example, you might create and use the following slot types for the OrderPizza intent:
 - Size – With enumeration values Small, Medium, and Large.
 - Crust – With enumeration values Thick and Thin.

Amazon Lex also provides built-in slot types. For example, AMAZON.NUMBER is a built-in slot type that you can use for the number of pizzas ordered. For more information, see Built-in Intents and Slot Types.

Currently, Amazon Lex supports only US English language. That is, Amazon Lex trains your bots to understand only US English.

(Source: <https://docs.aws.amazon.com/lex/latest/dg/what-is.html>)

Build your first chatbot

1) Login to AWS portal <https://signin.aws.amazon.com/>

The screenshot shows the official AWS website. At the top, there's a navigation bar with links for Contact Sales, Support, English, My Account, and a prominent orange button labeled "Create an AWS Account". Below the navigation, a dark banner features the text "One-Click Training and Deployment" and "Build, train, and deploy machine learning models quickly with Amazon Sagemaker". A "Learn more" link is provided. To the right of the text is a graphic of a neural network structure with green arrows indicating data flow. Below the banner is a horizontal row of four small icons: a robot, a water droplet, a rocket launching, and a database with binary code.

2) Click on “create an AWS Account”. Sign in with your credentials or create a new account.

The screenshot shows the AWS "Sign in" page. It has a "Sign in" button at the top left. Below it is a field for "Email address of your AWS account" with a placeholder "Or to sign in as an IAM user, enter your account ID or account alias instead." A "Next" button is located below the input field. To the right, there's a promotional section for free tier access. It features an illustration of a document with a checkmark. The text reads: "AWS Accounts Include 12 Months of Free Tier Access" and "Including use of Amazon EC2, Amazon S3, and Amazon DynamoDB". A link to "aws.amazon.com/free" for full offer terms is also present.

3) Once logged in, click on Amazon Lex to start creating your bot.

4) Click on create and create a new bot.

5) Once you click the create button, you will see a window like this. Click on the “custom bot” to make your own bot. The other bots are example bots and you can go through them to get a glance on some higher details.

Create your bot

Amazon Lex enables any developer to build conversational chatbots quickly and easily. With Amazon Lex, no deep learning expertise is necessary—you just specify the basic conversational flow directly from the console, and then Amazon Lex manages the dialogue and dynamically adjusts the response. To get started, you can choose one of the sample bots provided below or build a new custom bot from scratch.

6) Fill up all the details and create on the create button.

Custom bot BookTrip OrderFlowers ScheduleAppointment

Bot name

Language English (US)

Output voice ▾

Session timeout min ▾ i

Sentiment analysis Yes No i

IAM role AWSServiceRoleForLexBots i
Automatically created on your behalf

COPPA Please indicate if your use of this bot is subject to the [Children's Online Privacy Protection Act \(COPPA\)](#). Learn more i
 Yes No

Cancel Create

7) We have create a bot as follows:

Bot name iNeuronCourseDetails

Description

Language English (US)

Output voice ▾

Session timeout min ▾ i

Sentiment analysis Yes No i

IAM role AWSServiceRoleForLexBots i
Automatically created on your behalf

COPPA Please indicate if your use of this bot is subject to the [Children's Online Privacy Protection Act \(COPPA\)](#). Learn more i
 Yes No

8) We will start by creating intents for our bot. Click on the plus button next to intents.

The screenshot shows the 'iNeuronCourseDetails' bot interface. At the top, there are tabs for 'Editor', 'Settings', 'Channels', and 'Monitoring'. The 'Editor' tab is selected. Below the tabs, the word 'Intents' is followed by a blue plus sign button. A modal window titled 'Create intent' is open, prompting the user to 'Give a unique name for the new intent' with an example like 'TodaysWeather, GetDrinkOrder'. At the bottom of the modal are 'Previous' and 'Add' buttons. In the background, there are other tabs labeled 'and', 'o b', and 'o'.

We are going to create a “Welcome” intent which will greet the user on hearing/reading the defined utterances.

The screenshot shows the 'Editor' tab of the bot interface. On the left, under 'Intents', there is a list: 'Close', 'courseDetails', 'internshipOpportunities', 'interviewPreparation', 'job_assistance', and 'Welcome'. The 'Welcome' intent is selected and highlighted with an orange border. To the right of the list, there is a 'Slot types' section stating 'No slots created' and an 'Error Handling' section. On the right side, the 'Welcome' intent is expanded, showing 'Sample utterances'. It lists several examples: 'e.g. I would like to book a flight.', 'hey', 'what's up', 'hello there', 'hello', and 'hi'. Each example has a blue plus sign button to its right and a small 'x' icon at the end of the input field.

The screenshot shows the 'Response' section of an intent configuration. It includes a preview window displaying two message options: 'e.g. Thank you. Your {Drink_Name} has been ordered.' and 'Hi,Welcome to iNeuron!May i know how can i assist you?'. The second message is highlighted with a black rectangle. Below the preview is an 'Add Message' button.

We are not filling the slots for this intent and keeping the “Fullfillment” as return parameters to client. We will see the use of slots and use of lambda function for the next intent.

In the response section, we are giving the response that the bot is going to give after hearing all the utterances we have provided.

9) Similarly we are creating new intents as per our requirements that we need the bot to respond to.

The screenshot shows the configuration for the 'internshipOpportunities' intent. The left sidebar lists other intents like 'Close', 'courseDetails', 'interviewPreparation', 'job_assistance', and 'Welcome'. The right panel shows sample utterances: 'e.g. I would like to book a flight.', 'internship opportunity', 'how about internship', 'Can I get an internship after the course completion', 'Do you provide internship opportunities', and 'What are the internship opportunities'. Below this are sections for 'Lambda initialization and validation' and 'Slots'.

▼ Response ⓘ

Preview

|| ● Message ○ Custom Markup ⓘ ⚡

One of these messages will be presented at random.

e.g. Thank you. Your {Drink_Name} has been ordered. +

Both paid and unpaid internships are available with us. The sele ⚡

[+ Add Message](#)

Enable response card

Wait for user reply
If the user says "no," the following message will be presented.

10) let's see the Fulfillment using lambda function. We are creating one intent with sample utterances and once our bot receives that intent, it will ask the user for its details like name, mobile number and email. Then it will send a mail to the user with the course details attached and it will also send an email to the support team with all the details of the user requesting a callback.

Let's see how we are going to do it.

iNeuronCourseDetails Latest

Editor Settings Channels Monitoring

Intents + courseDetails Latest

Close courseDetails

internshipOpportunities

interviewPreparation

job_assistance

Welcome

Slot types +

No slots created

Error Handling

Sample utterances ⓘ

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

can you tell me about the course

about the course

i want to know about the course

i want the course details

what courses are available

i need the course information

course details

Once we have filled up the sample utterances. Let's fill the slots that we will be prompting the user for getting the user details.

In the slot name we give the values that we want from the user. The slot_type is the default validation parameter in Lex and we can use that to validate the values user gives against that slot. In the prompt we are going to prompt the user with the specific questions to get the slot values.

In the Fulfillment section, we are using Lambda function to send an email to the user on the given email id and to the support team with all the user details.

Let's see how to create the lambda function.

Go to the AWS console and click on “Lambda” inside the ”compute” section.

AWS Management Console

The screenshot shows the AWS Management Console sidebar. At the top is a search bar labeled "Find Services" with the placeholder "Example: Relational Database Service, database, RDS". Below it are sections for "Recently visited services" and "All services". The "Lambda" service is highlighted with a black rectangle.

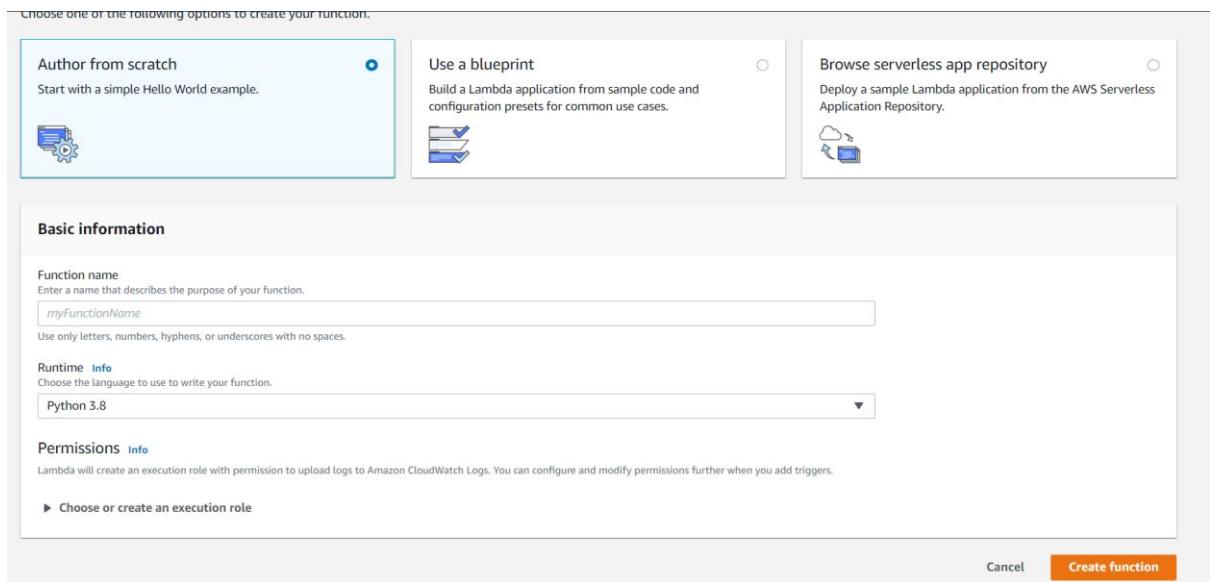
- Recently visited services:** Amazon Lex, Lambda, Billing, API Gateway
- All services:**
 - Compute:** EC2, Lightsail, ECR, ECS, Fargate, Lambda (highlighted), Batch, Elastic Beanstalk, Serverless Application Repository, AWS Outposts, EC2 Image Builder
 - Satellite:** Ground Station
 - Quantum Technologies:** Amazon Braket
 - Management & Governance:** AWS Organizations, CloudWatch, AWS Auto Scaling, CloudFormation, CloudTrail
 - Security, Identity, & Compliance:** IAM, Resource Access Manager, Cognito, Secrets Manager, GuardDuty, Inspector, Amazon Macie, AWS Single Sign-On, Certificate Manager, Key Management Service, CloudHSM

Click on the “create function” tab.

The screenshot shows the AWS Lambda "Functions" page. The left sidebar has "Functions" selected. The main area shows a table with three rows, each representing a function. The "Create function" button is highlighted with a red rectangle.

Function name	Description	Runtime	Code size	Last modified
function1		python 3.8	1.2 MB	2023-08-15
function2		node.js 18.x	1.2 MB	2023-08-15

Given the name of the function and select the runtime as “python”, you can select other runtimes as well. Click on the create function.



We have created a function named “sendEmail”. After clicking the “create function” button, you will see a page like this :

The screenshot shows the configuration page for the 'sendEmail' function. At the top, there are tabs for 'Configuration' (selected), 'Permissions', and 'Monitoring'. Below the tabs are buttons for 'Throttle', 'Qualifiers ▾', 'Actions ▾', 'ok', '▼', 'Test', and 'Save'. The main area is titled 'Designer' and contains a box for the function name 'sendEmail' and a 'Layers' section with '(0)'. There are buttons '+ Add trigger' and '+ Add destination'. At the bottom is a 'Function code' section with a 'Info' link.

Go to the “Function code” section:

Code entry type: Edit code inline Runtime: Python 3.8 Handler: lambda_function.lambda_handler

```

1 | """
2 | Lexbot Lambda handler.
3 | """
4 | import json
5 | import urllib3
6 |
7 | http = urllib3.PoolManager()
8 |
9 | def sendEmail(name, mobile_number, email):
10 |     data = {"emailId":email,"name": name, "mobile_number" : mobile_number}
11 |     encoded_data = json.dumps(data).encode('utf-8')
12 |     response = http.request('POST','http://f96f2618.ngrok.io/sendmail',headers={'Content-Type': 'application/json'},body = encoded_data)
13 |     return response.data.decode('utf-8')
14 |
15 | def lambda_handler(event, context):
16 |     #print('received request: ' + str(event))
17 |     name = event['currentIntent']['slots'][0]['name']
18 |     mobile_number = event['currentIntent']['slots'][0]['mobile_number']
19 |     email = event['currentIntent']['slots'][0]['emailId']
20 |     result = sendEmail(name, mobile_number, email)
21 |     response = {
22 |         "dialogAction": {
23 |             "type": "Close",
24 |             "fulfillmentState": "Fulfilled",
25 |             "message": {
26 |                 "contentType": "SSML",
27 |                 "content": "{} Please let us know if we can assist you with any other information.".format(result)
28 |             },
29 |         }
30 |     }
31 |     #print('result = ' + str(response))
32 |     return response
33 |

```

The function with name “lambda_handler” is the function first called once an intent with fulfilment as lamda is called and the it is passed as the “event” argument in the function.

Code entry type: Edit code inline Runtime: Python 3.8 Handler: lambda_function.lambda_handler

```

1 | """
2 | Lexbot Lambda handler.
3 | """
4 | import json
5 | import urllib3
6 |
7 | http = urllib3.PoolManager()
8 |
9 | def sendEmail(name, mobile_number, email):
10 |     data = {"emailId":email,"name": name, "mobile_number" : mobile_number}
11 |     encoded_data = json.dumps(data).encode('utf-8')
12 |     response = http.request('POST','http://f96f2618.ngrok.io/sendmail',headers={'Content-Type': 'application/json'},body = encoded_data)
13 |     return response.data.decode('utf-8')
14 |
15 | def lambda_handler(event, context):
16 |     #print('received request: ' + str(event))
17 |     name = event['currentIntent']['slots'][0]['name']
18 |     mobile_number = event['currentIntent']['slots'][0]['mobile_number']
19 |     email = event['currentIntent']['slots'][0]['emailId']
20 |     result = sendEmail(name, mobile_number, email)
21 |     response = {
22 |         "dialogAction": {
23 |             "type": "Close",
24 |             "fulfillmentState": "Fulfilled",
25 |             "message": {
26 |                 "contentType": "SSML",
27 |                 "content": "{} Please let us know if we can assist you with any other information.".format(result)
28 |             },
29 |         }
30 |     }
31 |     #print('result = ' + str(response))
32 |     return response
33 |

```

From the “event” argument we can derive our values in the slots like below :

```

def lambda_handler(event, context):
    #print('received request: ' + str(event))
    name = event['currentIntent']['slots'][0]['name']
    mobile_number = event['currentIntent']['slots'][0]['mobile_number']
    email = event['currentIntent']['slots'][0]['emailId']

```

Once we have got the values in our slots for the intent, we are going to pass these values as an argument in an another function we have written to send the mails.

```

def lambda_handler(event, context):
    #print('received request: ' + str(event))
    name = event['currentIntent']['slots']['name']
    mobile_number = event['currentIntent']['slots']['mobile_number']
    email = event['currentIntent']['slots']['emailId']
    result = sendEmail(name, mobile_number, email)
    response = {

```

Let's see what does this function "sendEmail" does:

```

import urllib3

http = urllib3.PoolManager()

def sendEmail(name, mobile_number, email):
    data = {"emailId":email, "name": name, "mobile_number": mobile_number}
    encoded_data = json.dumps(data).encode('utf-8')
    response = http.request('POST', 'http://f96f2618.ngrok.io/sendmail', headers={'Content-Type': 'application/json'}, body = encoded_data)
    return response.data.decode('utf-8')

```

This function hits an "Api" using the post method. We pass all the user details in the api in a json format. The Api returns a message on successfully sending the mails.

We have written a python code and deployed it as an "Api" which we are directly calling through this "sendEmail" function. Let's see the code written for creating the "Api".

```

Project ▾          sendEmail.py
sendEmail D:\Project\sendEmail
  .ipynb_checkpoints
  venv
    ngrokexe
    projects.txt
    sendemail.ipynb
    sendEmail.py
External Libraries
Scratches and Consoles

1 import smtplib
2 from email.mime.multipart import MIMEMultipart
3 from email.mime.text import MIMEText
4 from email.mime.base import MIMEBase
5 from email import encoders
6 from wsgiref import simple_server
7 from flask import Flask, request
8 from flask import Response
9 import os
10 from flask_cors import CORS, cross_origin
11
12
13 os.putenv('LANG', 'en_US.UTF-8')
14 os.putenv('LC_ALL', 'en_US.UTF-8')
15
16 app = Flask(__name__)
17 CORS(app)
18
19
20
21 @app.route("/sendmail", methods=['POST'])
22 @cross_origin()
23 def sendEmail():
24     try:
25         if request.json['emailId'] is not None:
26
27             toaddr = request.json['emailId'] ## email id of the user
28             contact_addr = "*****@gmail.com" ## the email id of the support team
29             fromaddr = "*****@gmail.com" ## the email id from where we are going to send the mail
30             mobile_number = request.json['mobile_number']
31             name = request.json['name']
32

```

```
sendEmail.py <input>
34     # instance of MIME Multipart
35     msg = MIME Multipart()
36
37     # storing the senders email address
38     msg['From'] = fromaddr
39
40     # storing the receivers email address
41     msg['To'] = ",".join(toaddr)
42     # msg['To'] = toaddr
43
44     # storing the subject
45     msg['Subject'] = "Test bot email"
46
47     # string to store the body of the mail
48     body = "This will contain attachment"
49
50     # attach the body with the msg instance
51     msg.attach(MIMEText(body, 'plain'))
52
53     # open the file to be sent
54     filename = "projects.txt"
55     attachment = open(filename, "rb")
56
57     # instance of MIMEBase and named as p
58     p = MIMEBase('application', 'octet-stream')
59
60     # To change the payload into encoded form
61     p.set_payload((attachment).read())
62
63     # encode into base64
64     encoders.encode_base64(p)
65
66     p.add_header('Content-Disposition', "attachment; filename= %s" % filename)
67
```

```
# attach the instance 'p' to instance 'msg'
msg.attach(p)

# creates SMTP session
s = smtplib.SMTP('smtp.gmail.com', 587)

# start TLS for security
s.starttls()

# Authentication
s.login(fromaddr, "****password****")# give your password here

# Converts the Multipart msg into a string
text = msg.as_string()
# sending the mail
s.sendmail(fromaddr, toaddr, text)

# send mail with username , mobile number and email id to concerned team

# instance of MIME Multipart
msg1 = MIME Multipart()
# storing the subject
msg1['Subject'] = "Query For Course Details"

# string to store the body of the mail
body1 = "person with name {0} have some queries regarding our course details." \
        "Please reach to {0} at his mobile number {1} and email id {2}".format(name,mobile_number,toaddr)

# attach the body with the msg instance
msg1.attach(MIMEText(body1, 'plain'))

text = msg1.as_string()
```

```

        text = msg1.as_string()
        # sending the mail
        s.sendmail(fromaddr, contact_add, text)

        # terminating the session
        s.quit()

    return Response("Course details is sent to the mail id %s" % toaddr +"\t" +
                    "and your details have been forwarded to the concerned team.")

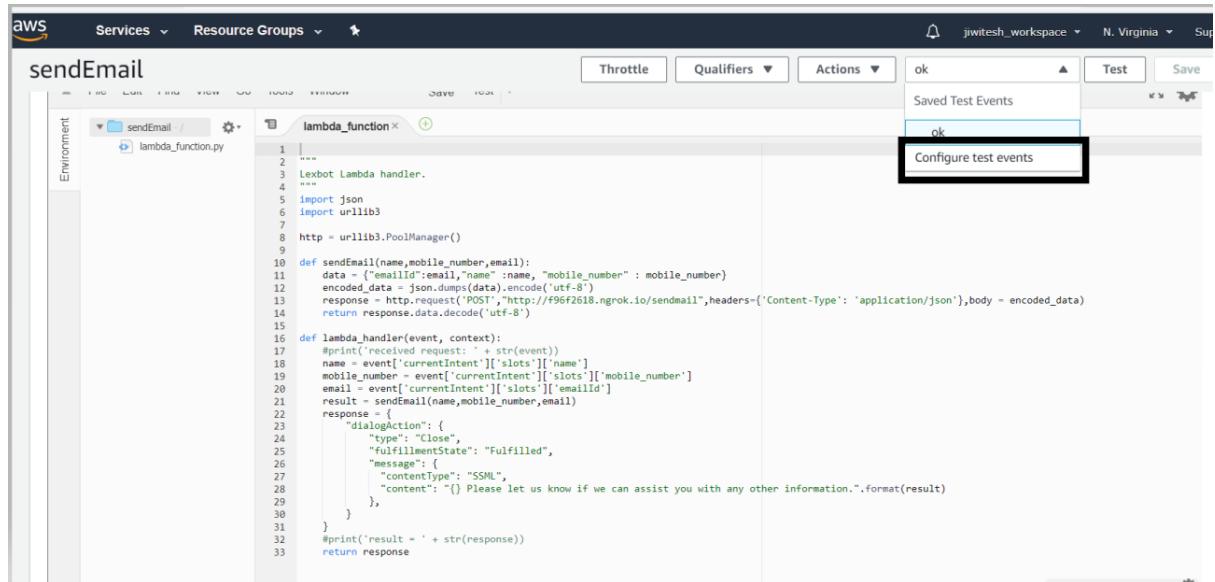
except ValueError:
    return Response("Error Occurred! %s" %ValueError)
except KeyError:
    return Response("Error Occurred! %s" %KeyError)
except Exception as e:
    return Response("Error Occurred! %s" %e)

#port = int(os.getenv("PORT"))
if name == "main":
    host = '0.0.0.0'
    port = 5000
    httpd = simple_server.make_server(host, port, app)
    print("Serving on %s %d" % (host, port))
    httpd.serve_forever()

```

We are keeping the Lambda function as simple as possible and we will not add any more functionality. Let's test our function written in the lambda.

For running the test, first save our lambda function, then click on the “configure test events”



You will see fields like this, create a new test event and set default values for the slots for testing the lambda function:

Saved test event

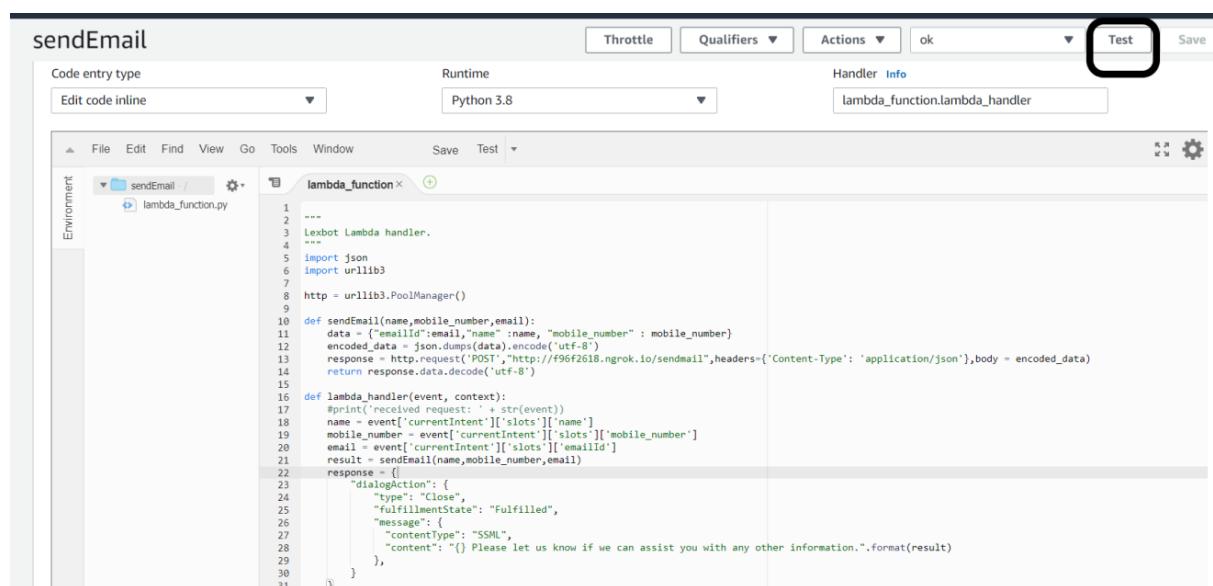
```

ok ▾
1 { "messageVersion": "1.0",
2   "invocationSource": "DialogCodeHook",
3   "userId": "John",
4   "sessionAttributes": {},
5   "bot": {
6     "name": "sendEmail",
7     "alias": "$LATEST",
8     "version": "$LATEST"
9   },
10  "outputDialogMode": "Text",
11  "currentIntent": {
12    "name": "courseDetails",
13    "slots": {
14      "name": "Raj",
15      "mobile_number": "8956785432",
16      "emailId": [REDACTED]@gmail.com"
17    },
18    "confirmationStatus": "None"
19  }
20}
21

```

Delete **Cancel** **Save**

Once the default values are set. Let's test our bot. Click on the test bot.



The screenshot shows the AWS Lambda function configuration interface for the 'sendEmail' function. At the top, there are tabs for 'Handler' and 'Info'. Below the tabs, there are dropdown menus for 'Code entry type' (set to 'Edit code inline'), 'Runtime' (set to 'Python 3.8'), and 'Handler' (set to 'lambda_function.lambda_handler'). On the right side, there are buttons for 'Throttle', 'Qualifiers', 'Actions', 'ok', 'Test' (which is highlighted with a black oval), and 'Save'. The main area contains the Lambda function code:

```

sendEmail
Code entry type: Edit code inline
Runtime: Python 3.8
Handler: lambda_function.lambda_handler
Actions: ok Test Save

File Edit Find View Go Tools Window Save Test
Environment: sendEmail / lambda_function.py
lambda_function.x
1 """
2 Lexbot Lambda handler.
3 """
4
5 import json
6 import urllib3
7
8 http = urllib3.PoolManager()
9
10 def sendEmail(name,mobile_number,email):
11     data = {'emailId':email,'name':name, 'mobile_number' : mobile_number}
12     encoded_data = json.dumps(data).encode('utf-8')
13     response = http.request('POST','http://f96f2618.ngrok.io/sendmail',headers={'Content-Type': 'application/json'},body = encoded_data)
14     return response.data.decode('utf-8')
15
16 def lambda_handler(event, context):
17     #print("received request: " + str(event))
18     name = event['currentIntent']['slots'][0]['name']
19     mobile_number = event['currentIntent']['slots'][0]['mobile_number']
20     email = event['currentIntent']['slots'][0]['emailId']
21     result = sendEmail(name,mobile_number,email)
22     response = [
23         {
24             "dialogAction": {
25                 "type": "Close",
26                 "fulfillmentState": "Fulfilled",
27                 "message": {
28                     "contentType": "SSML",
29                     "content": "{} Please let us know if we can assist you with any other information.".format(result)
30                 },
31             }
32         }
33     ]
34
35     return response

```

You will see a success message if your test works like this:

```

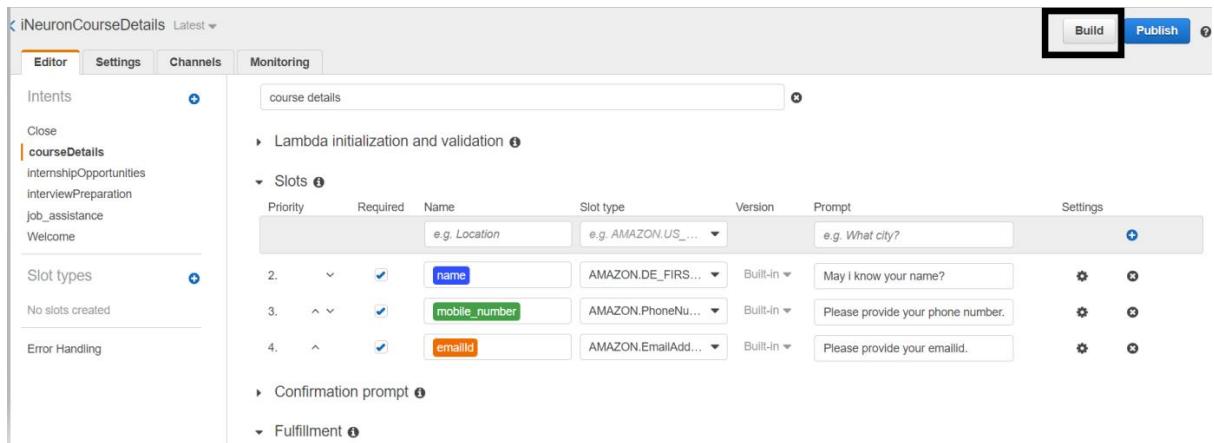
16 def lambda_handler(event, context):
17     #print('Received request: ' + str(event))
18     name = event['currentIntent']['slots'][name]
19     mobile_number = event['currentIntent']['slots'][mobile_number]
20     email = event['currentIntent']['slots'][emailId]
21     result = sendEmail(name, mobile_number, email)
22
23     response = {
24         "dialogAction": {
25             "type": "Close",
26             "fulfillmentState": "Fulfilled",
27             "message": {
28                 "contentType": "SSML",
29                 "content": "{} Please let us know if we can assist you with any other information.".format(result)
30             }
31         }
32     }

```

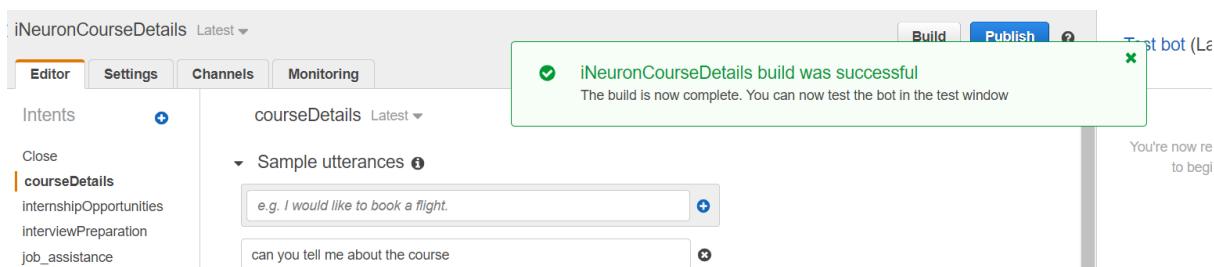
You will receive an error status if the function doesn't work properly.

Now that we have created all of our intents. You can test your bot with the intents by “building” your bot.

Click on the “build” button to build the bot:



Wait till you get the message that your bot is ready for testing.



Let's test our bot in the “Test Bot” window on the right.

The screenshot shows the Amazon Lex console interface. On the left, there's a sidebar with 'Channels' and 'Monitoring' tabs. The main area has a 'courseDetails' channel selected. Under 'Sample utterances', several examples are listed:

- e.g. I would like to book a flight.
- can you tell me about the course
- about the course
- I want to know about the course
- I want the course details
- what courses are available
- I need the course information

To the right, a chat window shows a conversation between a user ('hi') and the bot ('Hi, Welcome to iNeuron! May I know how can I assist you?'). Below the chat is a message input field ('Chat with your bot...'). Further down, there's an 'Inspect response' section with 'Dialog State: Failed' and a summary/detail toggle.

The second part of the screenshot shows a similar interface for the 'Channels' tab, where the same sample utterances are listed. To the right, a chat window shows a user asking 'I want to know about job opportunities' and the bot responding with 'We'll provide referral to the candidates. We have our own custom portal where you can search for jobs. The link is: https://jobs.ineuron.ai'. There's also a message input field and an 'Inspect response' section.

Now that we have built our bot and it's working fine. Let's go ahead with publishing the bot.

Before publishing the bot, go the settings tab and give an alias name to the bot. Again, build your bot.

The screenshot shows the AWS Lambda console with the 'iNeuronCourseDetails' function selected. In the left sidebar, the 'Settings' tab is active. The 'Aliases' section is expanded, showing:

- Alias name:** e.g., Prod
- Bot version:** Version 2

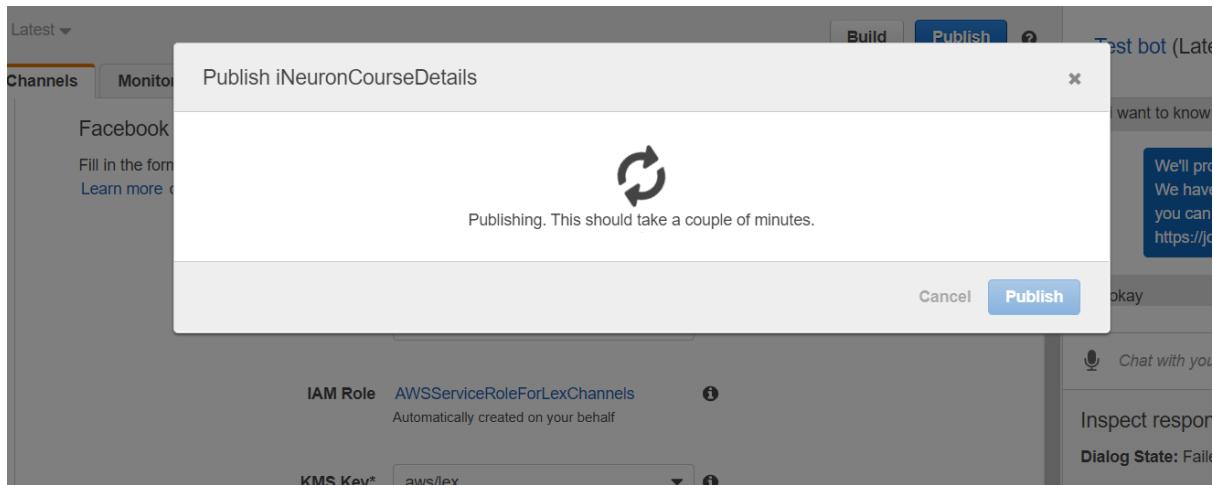
We will talk about deployment after publishing the bot, we will use this channel tab then.

The screenshot shows the AWS Lambda function configuration interface. The top navigation bar includes the AWS logo, Services dropdown, Resource Groups dropdown, and a star icon. Below the navigation is a breadcrumb trail: < iNeuronCourseDetails Latest. To the right are 'Build' and 'Publish' buttons. The main tabs are Editor, Settings, Channels (which is selected and highlighted in orange), and Monitoring. On the left sidebar under 'Channels', 'Facebook' is selected, with sub-options Kik, Slack, and Twilio SMS listed below it. The main content area is titled 'Facebook' and contains instructions: 'Fill in the form below and click activate to get a callback URL to use with Facebook. You can generate multiple callback URLs.' It also links to 'Learn more' for steps to integrate with Facebook. Form fields include 'Channel Name*' (input field with an info icon), 'Channel Description' (input field with an info icon), 'IAM Role' (set to 'AWSServiceRoleForLexChannels' with an info icon), 'KMS Key*' (set to 'aws/lex' with a dropdown arrow and an info icon), and 'Alias*' (input field with a dropdown arrow and an info icon). At the bottom is a large 'Activate' button.

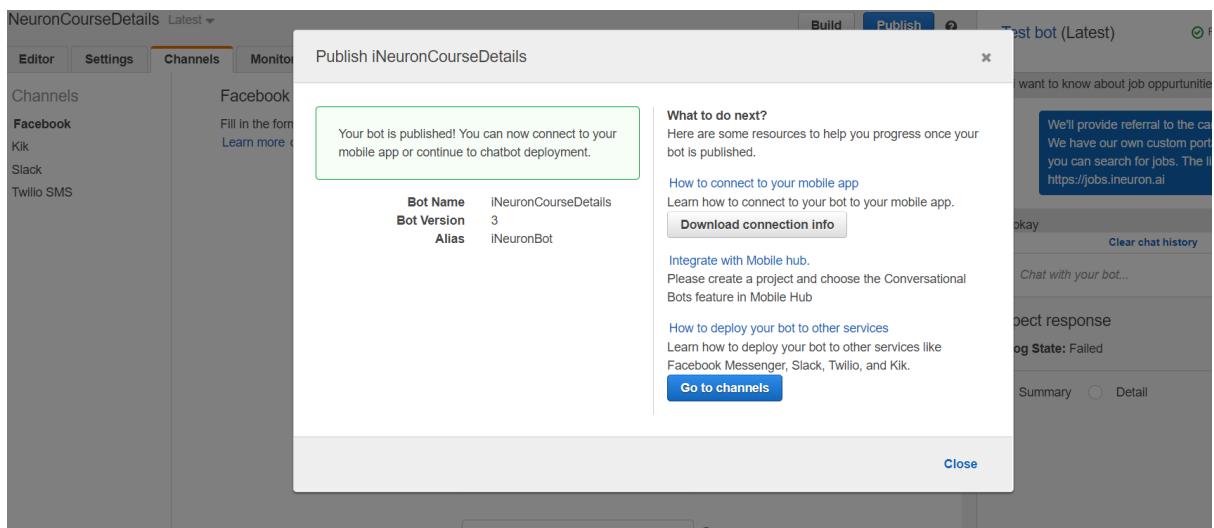
Click on the publish button and select the alias name that you have given.

The screenshot shows a modal dialog box titled 'Publish iNeuronCourseDetails'. The dialog has a close button 'x' at the top right. Inside, it says 'Publishing is the last step before you can connect your bot to your mobile app or chatbot.' A dropdown menu labeled 'Choose an alias' is set to 'iNeuronBot' with an info icon. Below it is a link 'Create a new alias'. At the bottom are 'Cancel' and 'Publish' buttons. The background of the main interface is dimmed, showing the same configuration screen as the first image, with the 'Channels' tab selected and the 'Facebook' integration setup.

Click on the publish button.



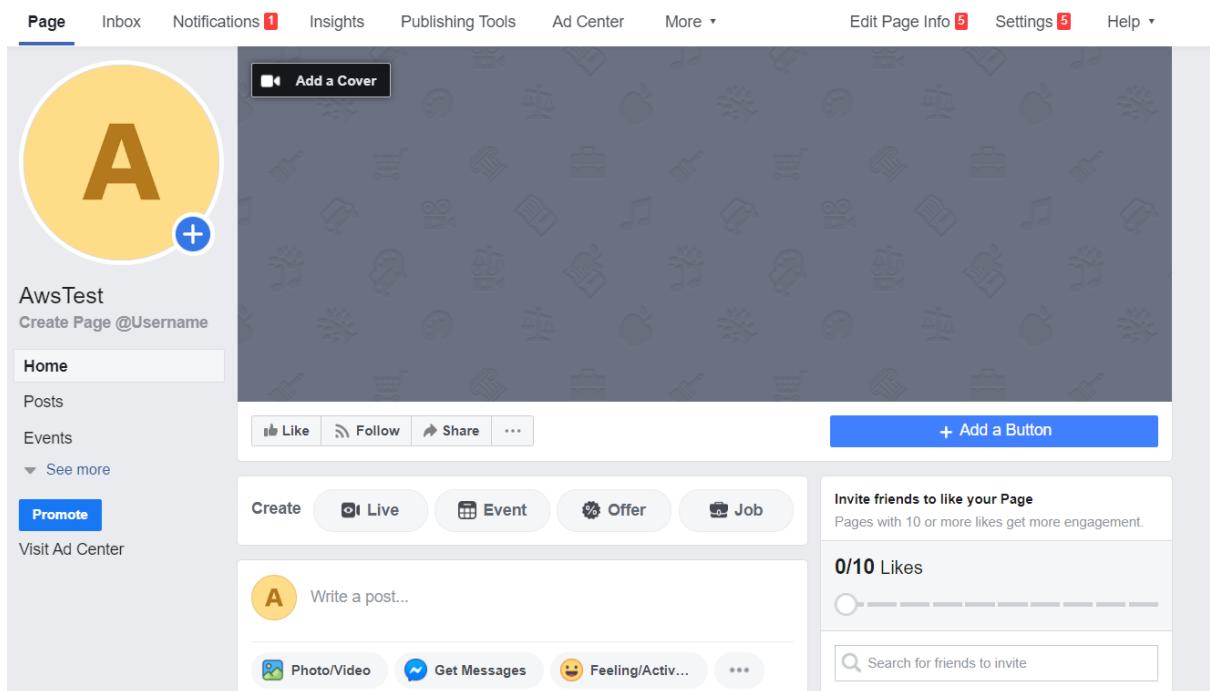
Now your bot is published. Let's deploy our bot on Facebook. Click on the “Go to Channels”.



DEPLOYMENT ON FACEBOOK

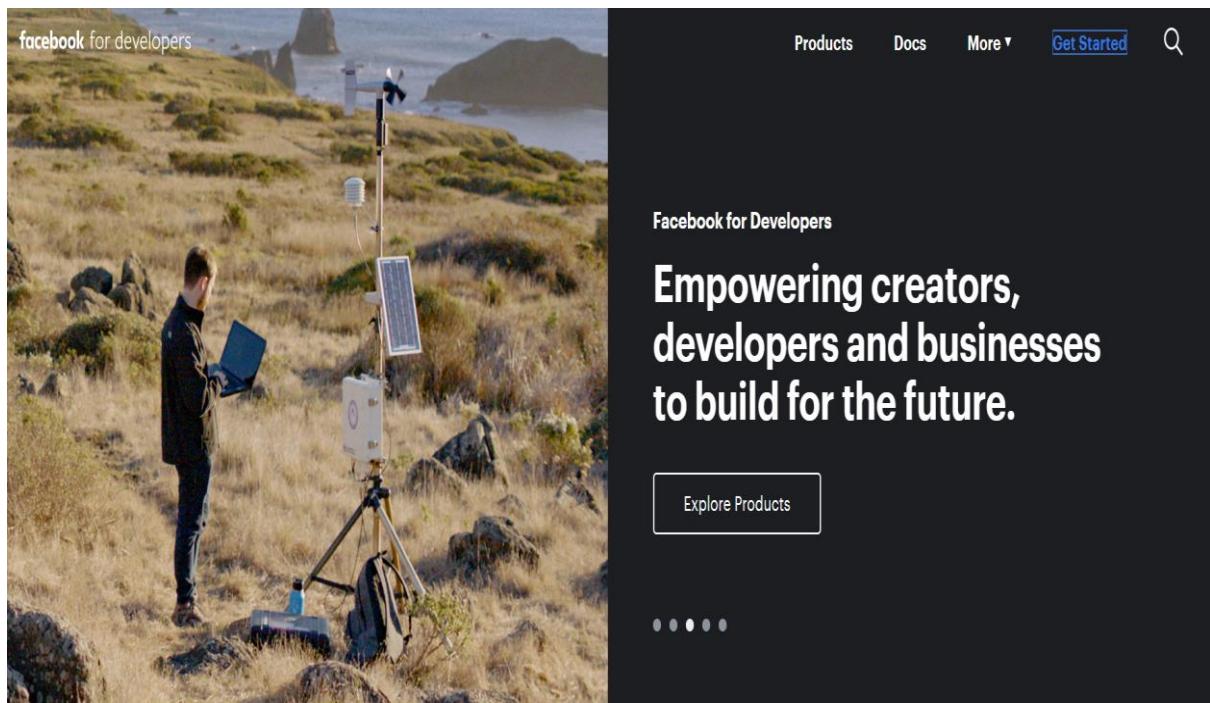
Go to Facebook and create a page on which we will be integrating our chatbot. We have “AwsTest” page created already on facebook.

Screenshot of a Facebook page named "AwsTest". The page cover photo is a yellow circle with a large letter "A" in the center. The page has 0 likes and 0/10 friends invited. It includes sections for "Create", "Live", "Event", "Offer", and "Job".



Go to Facebook for developers and click on Get started.

Screenshot of the Facebook for Developers homepage. It features a hero image of a person working with a laptop in a field next to a weather station. The main headline reads: "Empowering creators, developers and businesses to build for the future." A "Get Started" button is visible at the top right.



Changes to F8 2020

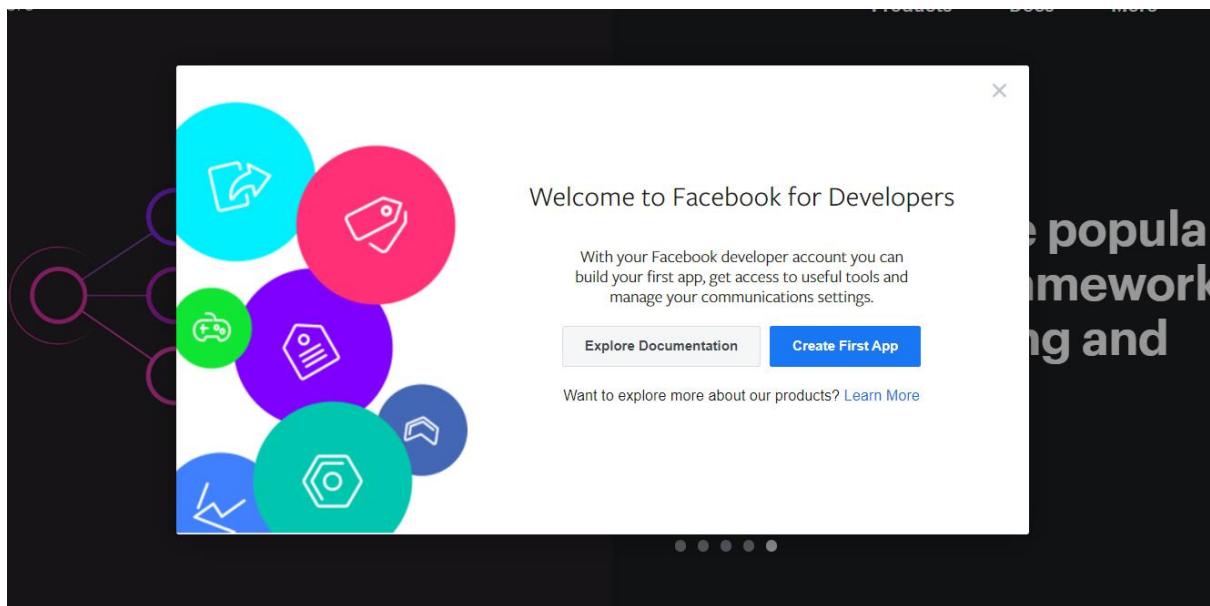
[Learn More](#)

Best Practices for Designing Great Messaging Experiences on Messenger

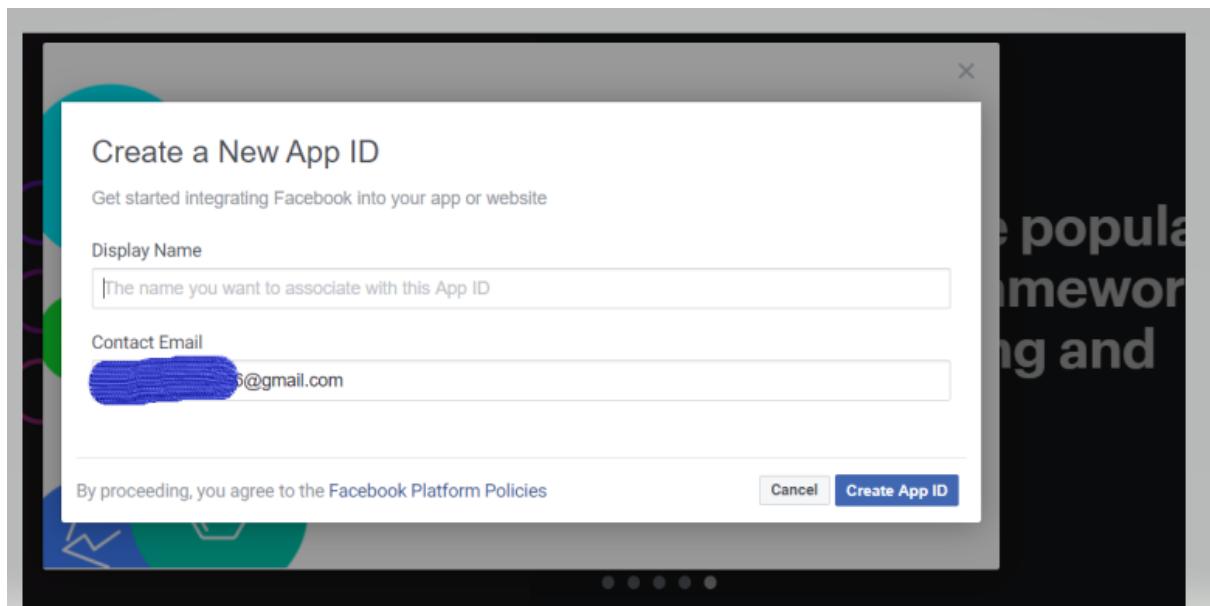
Messenger Launches One-Time Notification API to Enable Businesses to Send Important Follow-up

If you are a new user, create an account on facebook for developer and then proceed.

Click on “create first App”



Give a display name



Complete the security check:

Security Check

Please complete the Security Check.

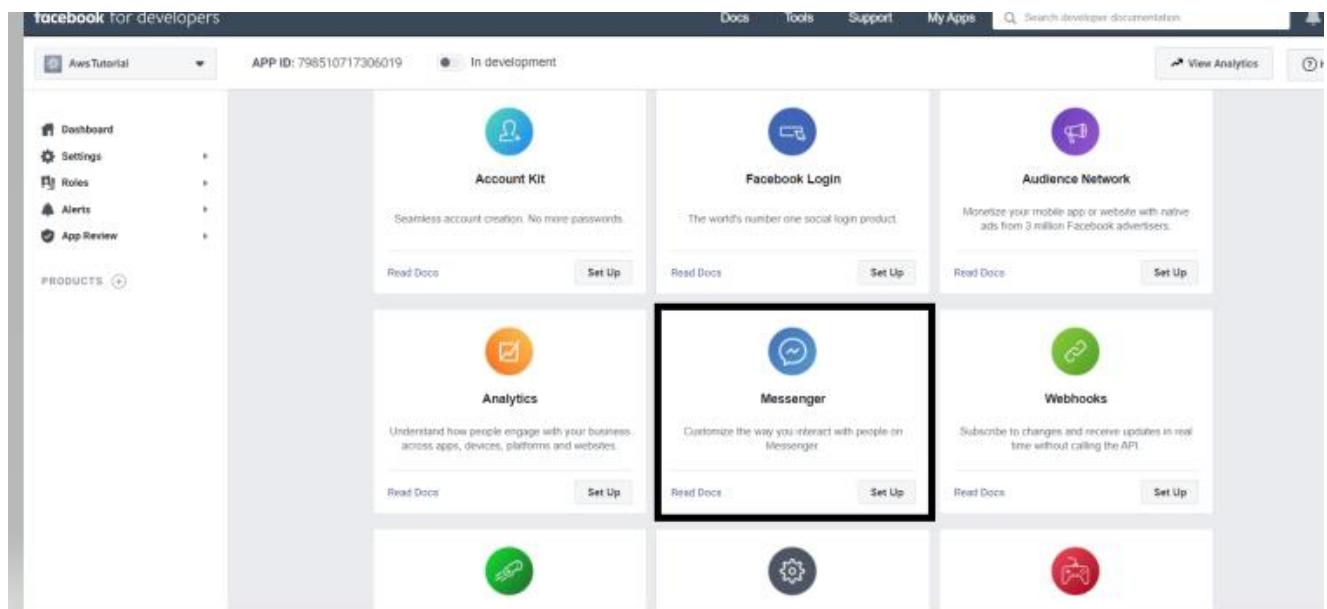
Select all images with **bridges**

Why am I seeing this?

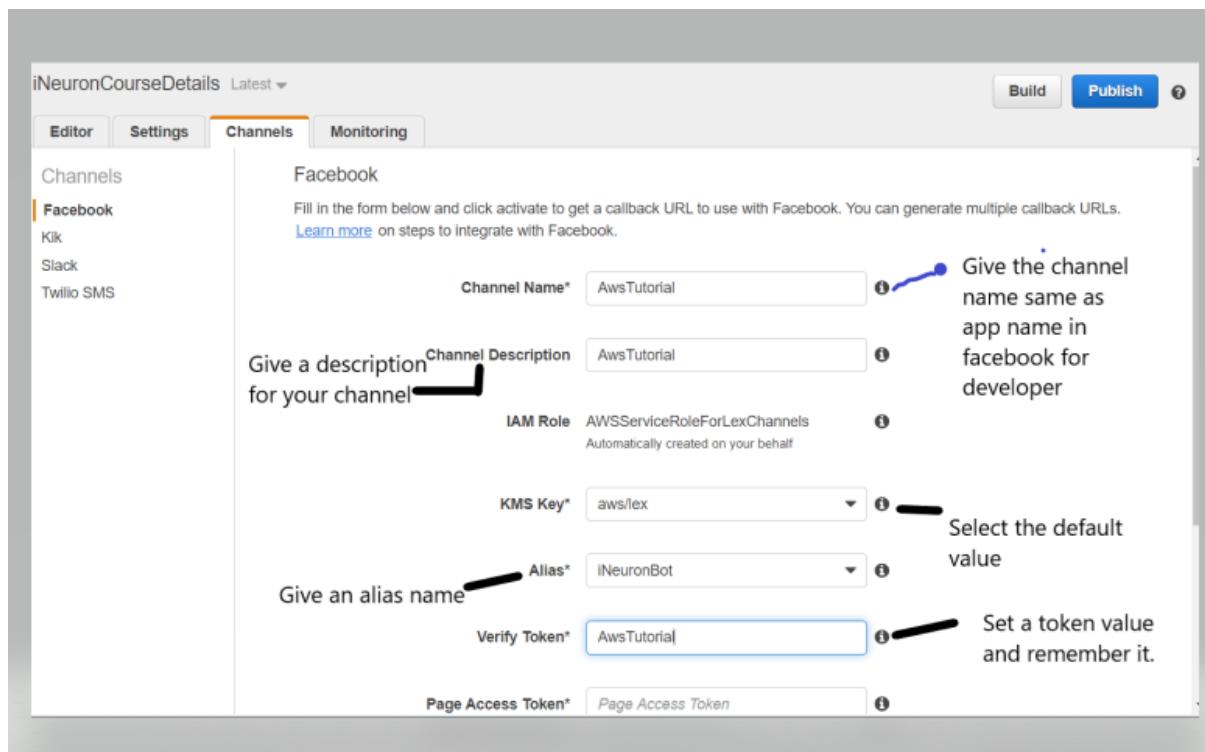
If you think this doesn't go against our Community Standards let us know.

Submit **Cancel**

You will see the below page once the app is created. Click on the messenger tab and then on the setup button:



Now go back to your Lex bot page and enter the details as shown below:



To get the page access token, go back to your app in Facebook for developers, click on settings below messenger and then scroll down to come to the access token section:

The screenshot shows the Facebook for Developers dashboard for an app with ID 798510717306019. The left sidebar has sections for Dashboard, Settings, Roles, Alerts, App Review, and Products (Messenger, Settings). The main content area displays the 'Access Tokens' section with instructions to generate a Page access token. It notes that the app is in development mode and lists requirements: being one of the page admins and having the app granted the page's permission to manage and access Page conversations in Messenger. A note states that in dev mode, tokens will only be accessible to people who manage the app or page. A button labeled 'Add or Remove Pages' is visible.

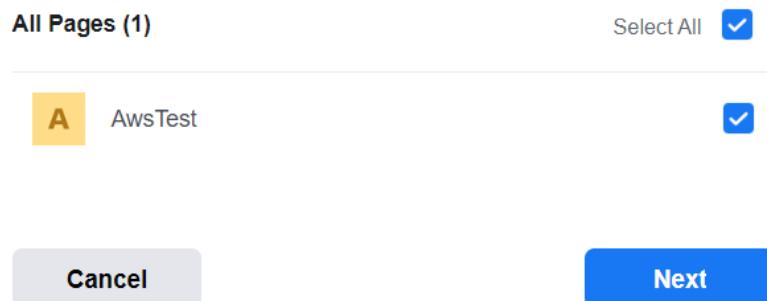
Click on the Add or remove pages button to add the page created on facebook. You will see a popup like this, press continue as:

The screenshot shows a Facebook login dialog box titled 'Log In With Facebook - Brave'. It asks 'Continue as [REDACTED]'. Below it, it says 'AwsTutorial will receive your name and profile picture. This doesn't let AwsTutorial post to Facebook without your permission.' There are 'Cancel' and 'Continue as [REDACTED]' buttons. At the bottom, it says 'Not Pranjal Sijaria? Log into another account.'

Select the app from the list :

What Pages do you want to use with AwsTutorial?

In the next step, you will determine what AwsTutorial can do with the Pages you selected.



You will get a successful message:

You've now linked AwsTutorial to Facebook

You can update what AwsTutorial can do in your [Business Integrations Settings](#). To finish setup, AwsTutorial may require additional steps.

OK

You will see the access token section changed like this :

Access Tokens	
Generate a Page access token to start using the platform APIs. You will be able to generate an access token for a Page if:	
<ol style="list-style-type: none">1. You are one of the Page admins, and2. The app has been granted the Page's permission to manage and access Page conversations in Messenger.	
Note: If your app is in dev mode, you can still generate a token but will only be able to access people who manage the app or Page.	
Pages ↑	Tokens
 AwsTest 102360038048777	Token Generated Generate Token
Add or Remove Pages (1)	

Click on the Generate Token. You will get a message like this:

A AwsTest
102360038048777

To protect your security, **ONLY share this token with app developers you trust.**

This token will only be shown once, so keep it safe. If it gets lost, you'll need to create a new one. Anyone could potentially use this token to impersonate this page, depending on the privacy settings of your app. If you wish to revoke all previously generated tokens from a page, you can remove this page from the app using the button below the table.

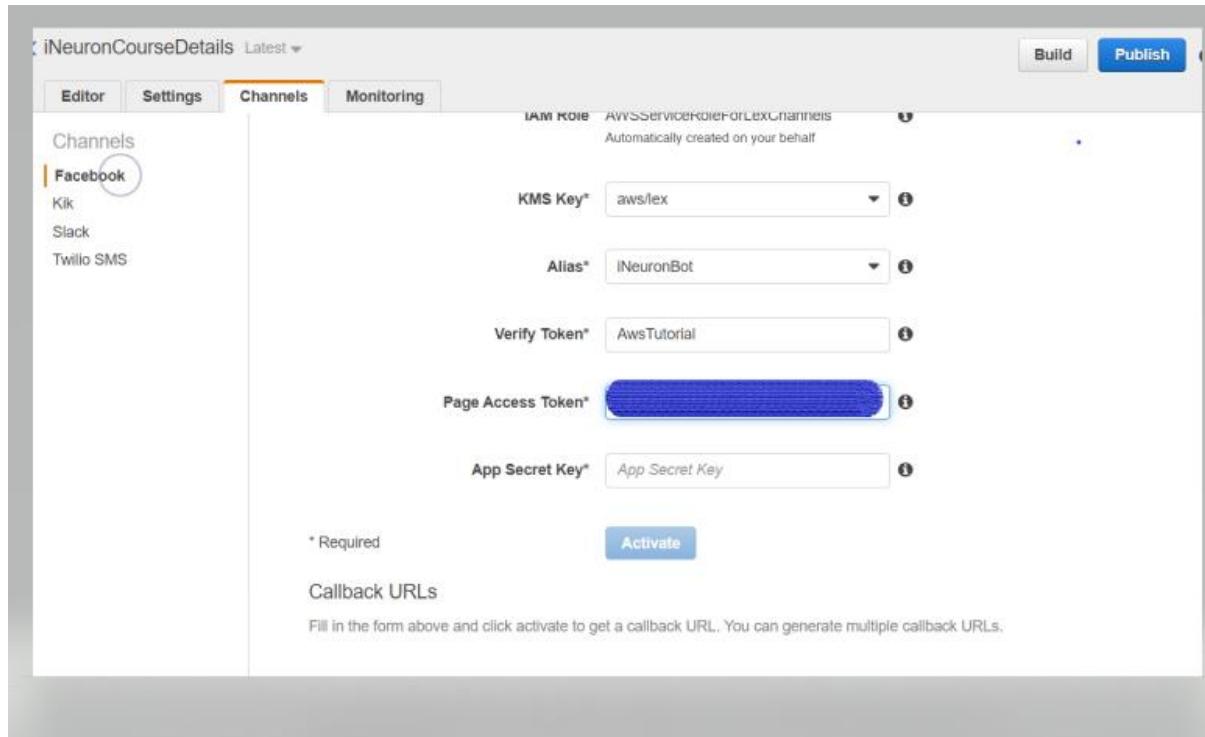
I Understand

EAALW.....  Copy

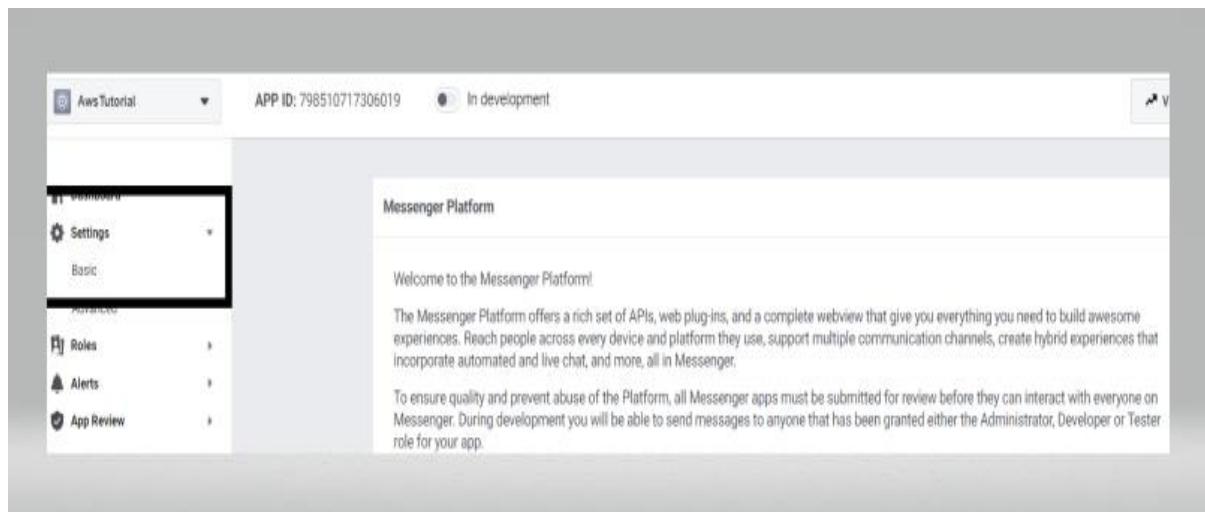
Done

Click on “I understand” and then copy the given code.

Paste the code to the page access token section of your Bot channel page:



Now for app secret key, go back to facebook for developers and click on basic tab below settings:



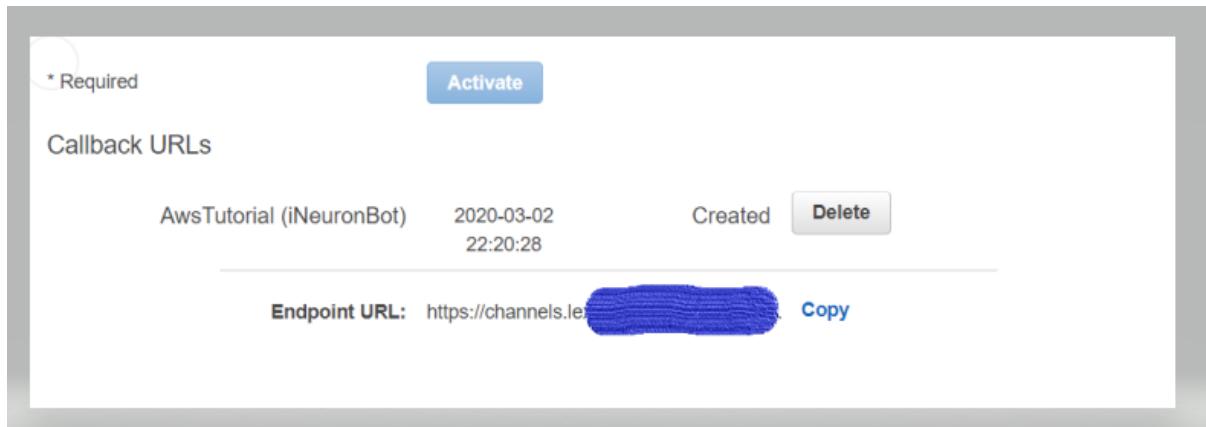
You will see a page like this after clicking. Click on the show button, you will be prompted to enter your password for facebook, enter your password and you can copy the code:

The screenshot shows the Facebook App Settings page under the 'Basic' tab. The 'App ID' is listed as 798510717306019. The 'App Secret' field is highlighted with a black rectangle and contains several asterisks. Below it, there are fields for 'Display Name' (AwsTutorial), 'Namespace', 'Contact Email' (pranjalijaria0406@gmail.com), 'Privacy Policy URL', 'Terms of Service URL', and 'Category'. A 'Show' button is located next to the App Secret field. At the bottom, there is a large redacted area labeled 'App Secret' with a 'Reset' button.

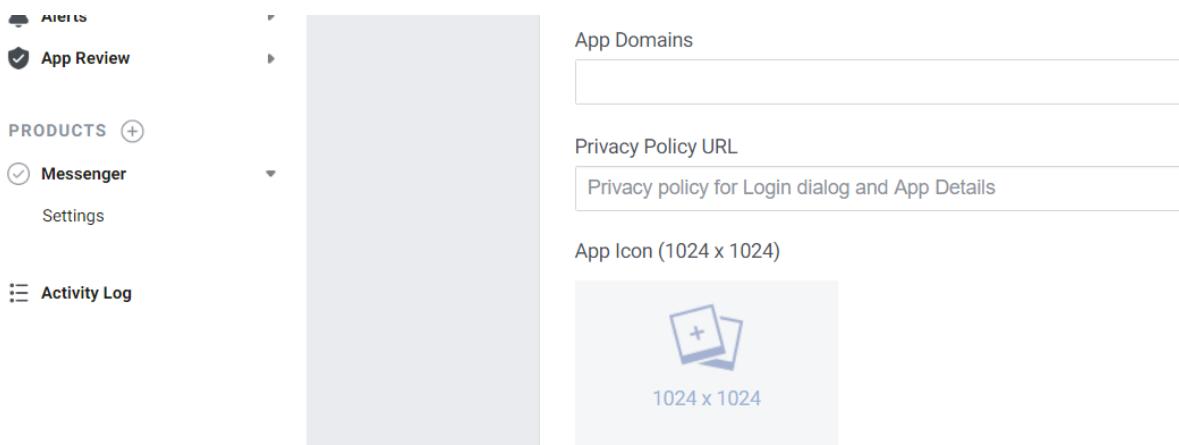
Copy the app secret and come back to your bot channel page and paste in the app secret field:

The screenshot shows the AWS Lambda function configuration page under the 'Channels' tab. It is configured for the Facebook channel. The 'KMS Key*' dropdown is set to 'aws/lex'. The 'Alias*' dropdown is set to 'iNeuronBot'. The 'Verify Token*' field contains 'AwsTutorial'. The 'Page Access Token*' field is redacted. The 'App Secret Key*' field contains '5da...' and is also redacted. At the bottom, there is a note '* Required' and a blue 'Activate' button.

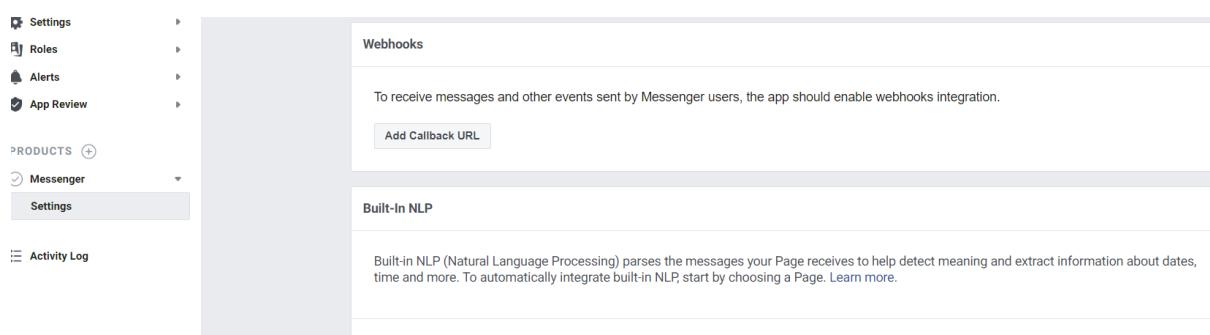
Click the activate button after filling all the details. You will get an endpoint url as shown below after activation. Copy the url:



Go back to facebook for developer page in messenger settings tab:

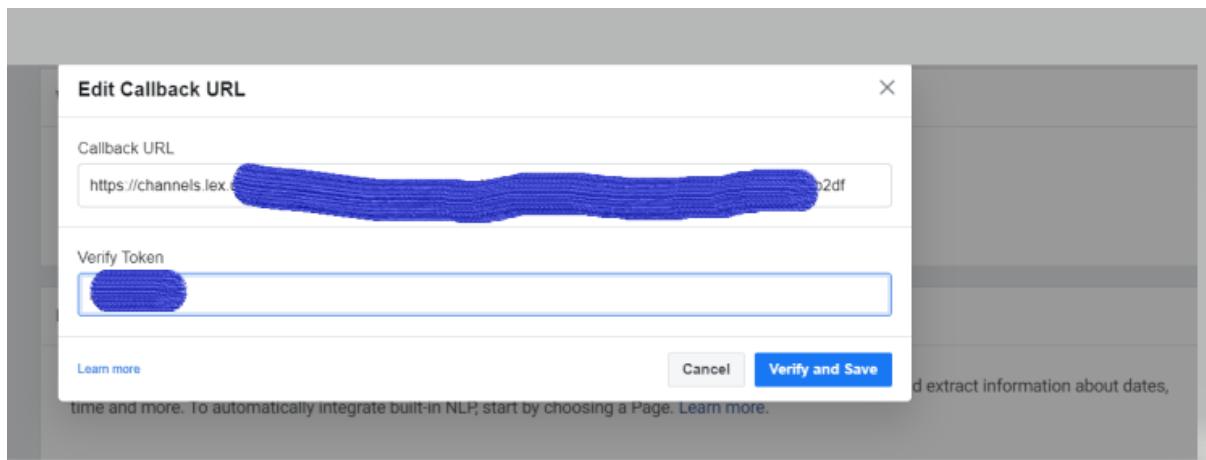


Click on add callback url under webhooks section:



Enter all the details, paste the url in the first box and in second field enter the token value created at the bot channel page:





After clicking Verify and Save, the webhook section will be updated and you can see an Add subscription button like below. Click on it.

Webhooks

To receive messages and other events sent by Messenger users, the app should enable webhooks integration.

Callback URL	Verify Token
<input type="text" value="https://channels.lex.us-east-1.amazonaws.com/facebook/webhook/c15d..."/>	<input type="text" value="....."/>
Validation requests and Webhook notifications for this object will be sent to this URL.	
Token that Facebook will echo back to you as part of callback URL verification.	

[Edit Callback URL](#)

[Show Recent Errors](#)

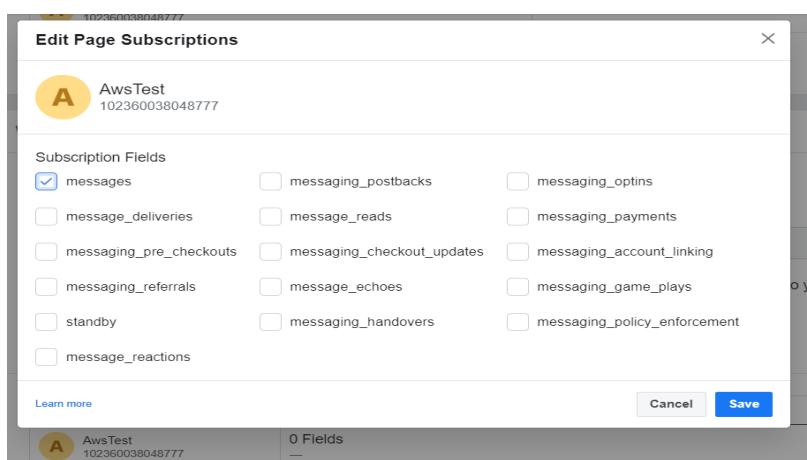
Pages ↑	Webhooks
AwsTest 102360038048777	0 Fields —

[Add Subscriptions](#)

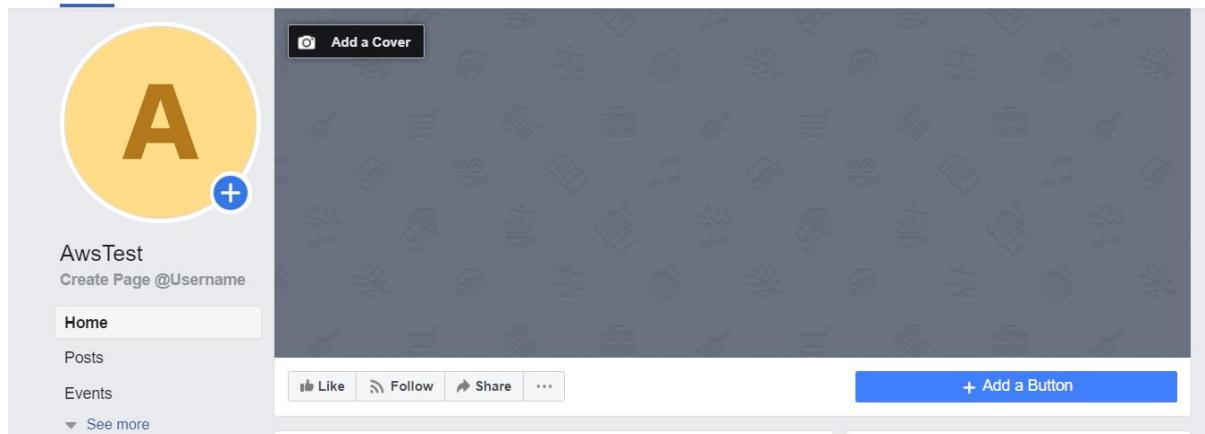
[Add or Remove Pages](#)



After clicking, check the messages box and save.



Now the setup is completed. Go to your page on facebook and hit refresh.



Click on Add a button.

Select the “send message” button in the “Contact you” tab and click next.

A screenshot of the "Add a Button to Your Page" configuration interface. At the top, it says "Add a Button to Your Page" and has a close button. Below that is a "Preview" section showing a dark grey background with social media icons and a "Send Message" button. Underneath the preview are the standard "Like", "Follow", "Share", and "More" buttons. The main configuration area is titled "Step 1: Which button do you want people to see?". It lists several options with checkboxes:

- Book with you
- Contact you (checkbox is checked)
- Contact Us
- Send Message (radio button is selected)
- Call Now
- Learn more about your business
- Shop with you
- Download your app or play your game

At the bottom of the configuration area, it says "Step 1 of 2" and has "Cancel" and "Next" buttons.

Click on messenger on the next page and click finish.

Add a Button to Your Page

Preview



Like Follow Share ... Send Message

✓ Your Button Send Message Edit

Step 2: Where would you like this button to send people?

When customers click your button, they will be directed to a place where they can take an action or find more information.

Messenger
You will receive messages from people in your Page Inbox.

Step 2 of 2 Back Finish

Add a Button to Your Page

Preview



Like Follow Share ... Send Message

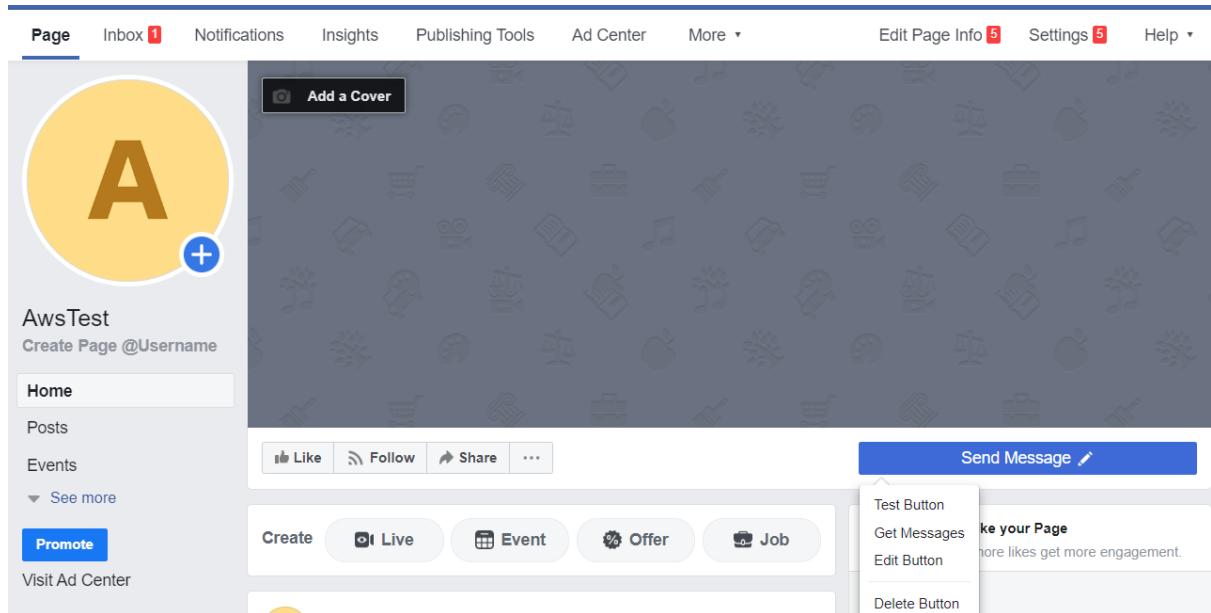
✓ Your Button Send Message Edit

✓ Configured Send Message Edit

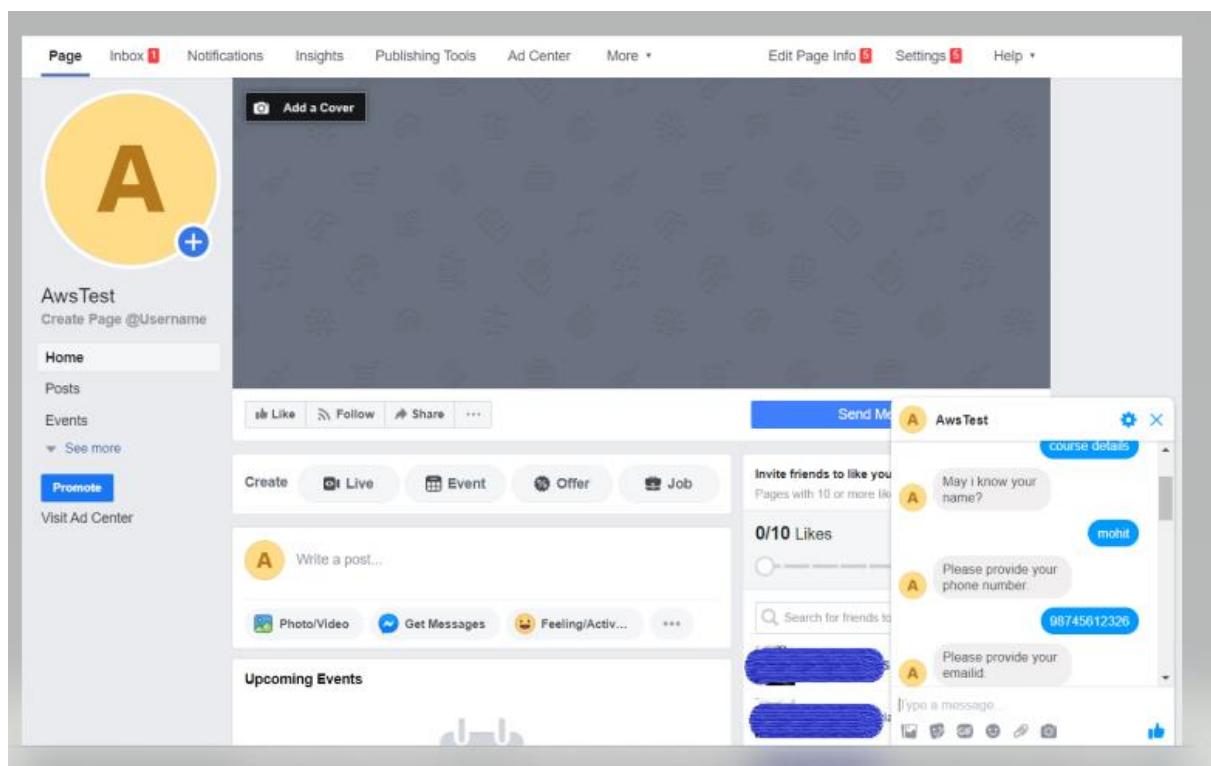
Final Review Back Finish

Now your chat bot is ready to be used.

Hover the mouse over send message button and click on test button:



Start chatting.



Aws Test

Please provide your emailid.

mohitkashyap3707@gmail.com

Course details is sent to the mail id mohitkashyap3707@gmail.com and your details have been forwarded to the concerned team.
Please let us know if

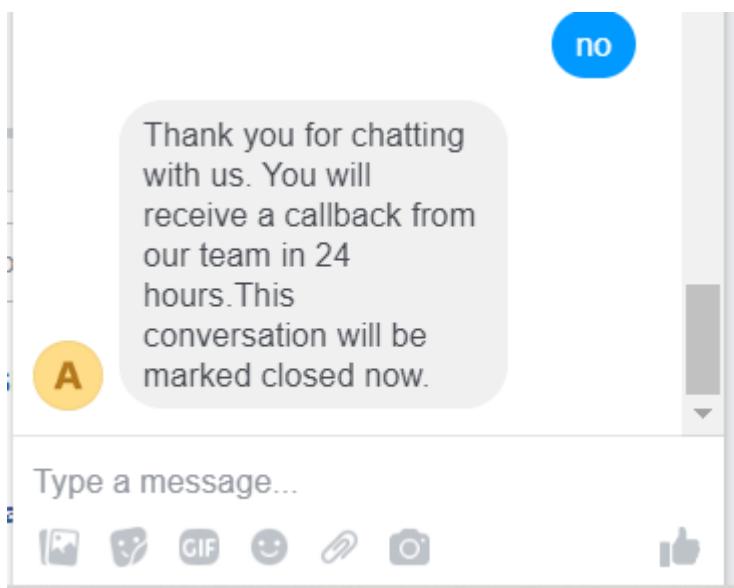
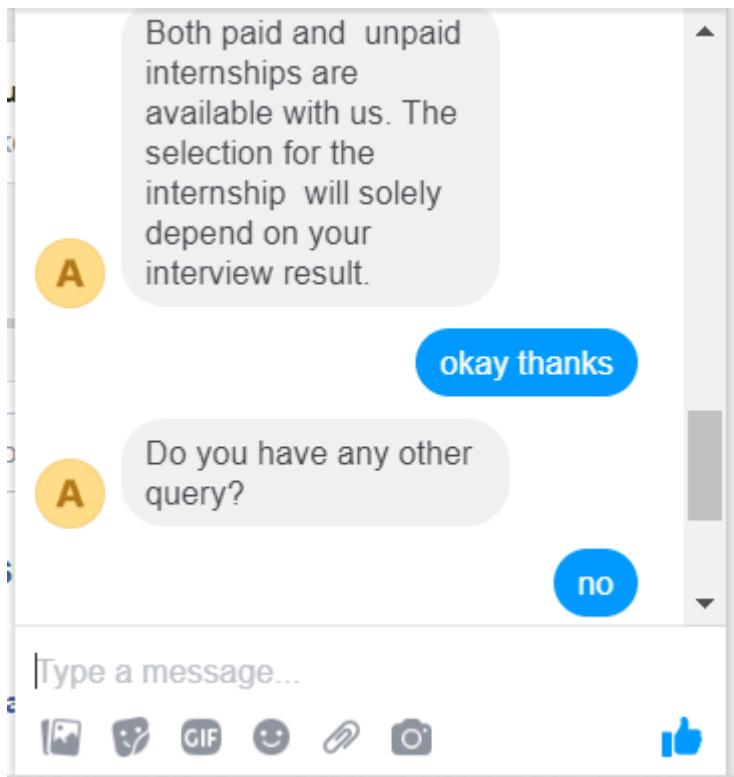
Type a message...

job assistance

We'll provide referral to the candidates. We have our own custom portal where you can search for jobs. The link is:
<https://jobs.ineuron.ai>

internship

Both paid and unpaid internships are



Now your Bot is deployed on facebook and is ready for use.

Happy Learning!!!!!!