### ****AI-Powered Cloud Chatbot****

### ****Project Overview****

We will create a **chatbot** that interacts with users, processes their queries, and responds based on a knowledge base stored in **DynamoDB**. The chatbot will be deployed on AWS and accessible via a web app or messaging platform like Telegram.

## ****Tech Stack****

* **AWS Lex** – NLP-based chatbot service
* **AWS Lambda** – Backend processing
* **DynamoDB** – Database for storing responses
* **API Gateway** – Exposing Lambda as an API
* **React.js (Optional)** – Frontend to interact with the chatbot

## ****Step-by-Step Implementation****

### ****Step 1: Set Up AWS Lex for NLP****

1. Go to **AWS Console → Amazon Lex**.
2. Create a new **Lex bot** (e.g., CloudChatBot).
3. Define **Intents** (categories of user queries). Examples:
   * GreetingIntent → Handles "Hello," "Hi," etc.
   * SupportIntent → Handles "How can I reset my password?"
   * FallbackIntent → Handles unknown queries.
4. Add **Utterances** (user inputs triggering an intent).
5. Define **Slots** (variables needed for responses).
6. Set **Responses** (static responses or API calls to Lambda).
7. Deploy and test the bot.

### ****Step 2: Create AWS Lambda Function for Dynamic Responses****

1. Go to **AWS Lambda → Create Function**.
2. Choose **Python 3.x** (or Node.js) as the runtime.
3. Write a function to process chatbot requests:

python

CopyEdit

import json

import boto3

dynamodb = boto3.resource("dynamodb")

table = dynamodb.Table("ChatbotResponses")

def lambda\_handler(event, context):

user\_message = event["inputTranscript"]

# Query the DynamoDB table for the response

response = table.get\_item(Key={"query": user\_message})

if "Item" in response:

bot\_reply = response["Item"]["response"]

else:

bot\_reply = "Sorry, I don't have an answer for that."

return {

"dialogAction": {

"type": "Close",

"fulfillmentState": "Fulfilled",

"message": {"contentType": "PlainText", "content": bot\_reply}

}

}

1. Deploy the Lambda function and copy its ARN.
2. Attach this Lambda function to the **AWS Lex intents** under "Fulfillment".

### ****Step 3: Store Responses in DynamoDB****

1. Go to **AWS DynamoDB → Create Table**.
2. Table Name: ChatbotResponses
3. Primary Key: query (String)
4. Add some predefined responses:

| query | response |
| --- | --- |
| "What is cloud computing?" | "Cloud computing delivers computing services over the internet." |
| "How to create an EC2 instance?" | "Go to AWS EC2, select 'Launch Instance,' choose an OS, and configure settings." |

### ****Step 4: Deploy API Gateway for Web Access****

1. Go to **AWS API Gateway → Create API**.
2. Choose **REST API**.
3. Create a **POST method** and link it to your **Lambda function**.
4. Deploy the API and copy the **API endpoint**.

### ****Step 5: (Optional) Create a Frontend Using React.js****

1. Set up a React.js project:

bash

CopyEdit

npx create-react-app chatbot-ui

cd chatbot-ui

npm start

1. Modify App.js to interact with the API Gateway:

javascript

CopyEdit

import React, { useState } from "react";

function App() {

const [message, setMessage] = useState("");

const [response, setResponse] = useState("");

const sendMessage = async () => {

const res = await fetch("YOUR\_API\_GATEWAY\_URL", {

method: "POST",

headers: { "Content-Type": "application/json" },

body: JSON.stringify({ userInput: message }),

});

const data = await res.json();

setResponse(data.reply);

};

return (

<div>

<h1>Cloud Chatbot</h1>

<input value={message} onChange={(e) => setMessage(e.target.value)} />

<button onClick={sendMessage}>Send</button>

<p>Bot: {response}</p>

</div>

);

}

export default App;

1. Run the app and interact with the chatbot!

## ****Final Outcome****

✅ A **cloud-based AI chatbot** deployed on AWS, accessible via API or frontend, dynamically responding based on user queries stored in DynamoDB.

Would you like help deploying it or integrating more features like voice support? 🚀