Building ChatBot Using AWS

Build On:

- Amazon S3
- Amazon Lex
- Amazon Lambda
- Amazon DynamoDB
- Amazon CloudFront
- Amazon API GateWay
- Amazon IAM

Introduction

This project is based on AWS and is intended to make a smart Lex bot being used on Web Applications.

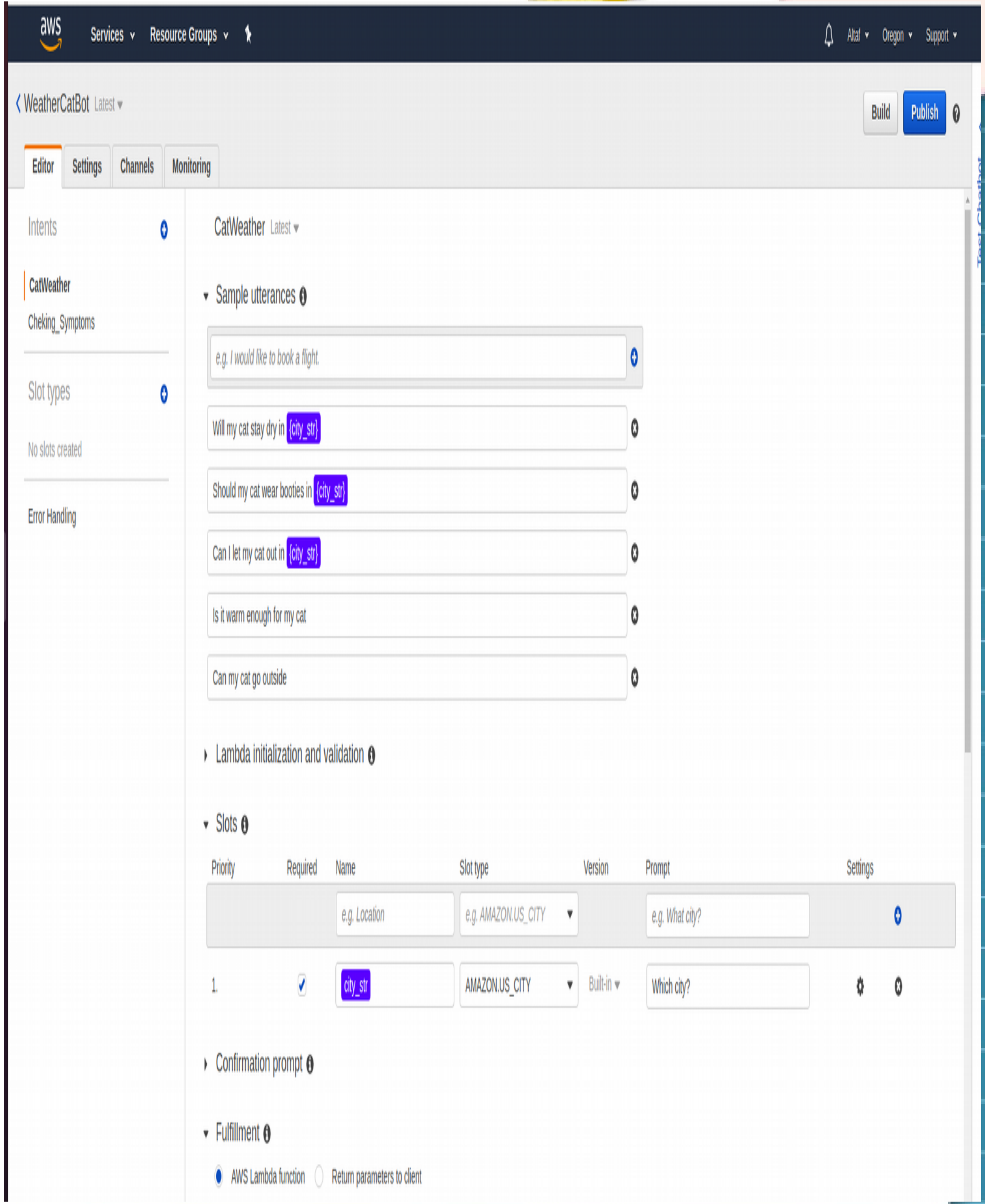
It uses many services of AWS to build and the purpose of this project is to take city data from the user and to return the temperature of the city and to predict whether it is ideal for their pets to go out. This is all based on speech recognition and text to speech using Amazon Lex.

It uses predefined values in the form of a CSV file which is being used to give the output of the temperature values. It uses a secured way of connecting to the user by redirecting HTTP to HTTPS and with rest endpoint with API gateway and then by creating serverless Lambda function for efficient and fast invocations.

It also uses DynamoDB for amazing NoSQL database performance. And most important used Lex for amazing voice recognition and text to speech.

Step1: Create a Simple Lex Bot

- Create a Custom Bot
- Add your Intent
- Create a slot to get what needed to get the desired output
- Test the bot with multiple inputs. And it is smart enough to give the desired output even though we haven't provide that intent.



WeatherCatBot Latest

Intents
- CatWeather
- Cheking_Symptoms

Slot types
- No slots created

Error Handling

CatWeather Latest

Sample utterances

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

- Will my cat stay dry in {city_str}
- Should my cat wear booties in {city_str}
- Can I let my cat out in {city_str}
- Is it warm enough for my cat
- Can my cat go outside

Lambda initialization and validation

Slots

Priority	Required	Name	Slot type	Version	Prompt	Settings
		e.g. Location	e.g. AMAZON.US_CITY		e.g. What city?	
1.	✓	city_str	AMAZON.US_CITY	Built-in	Which city?	

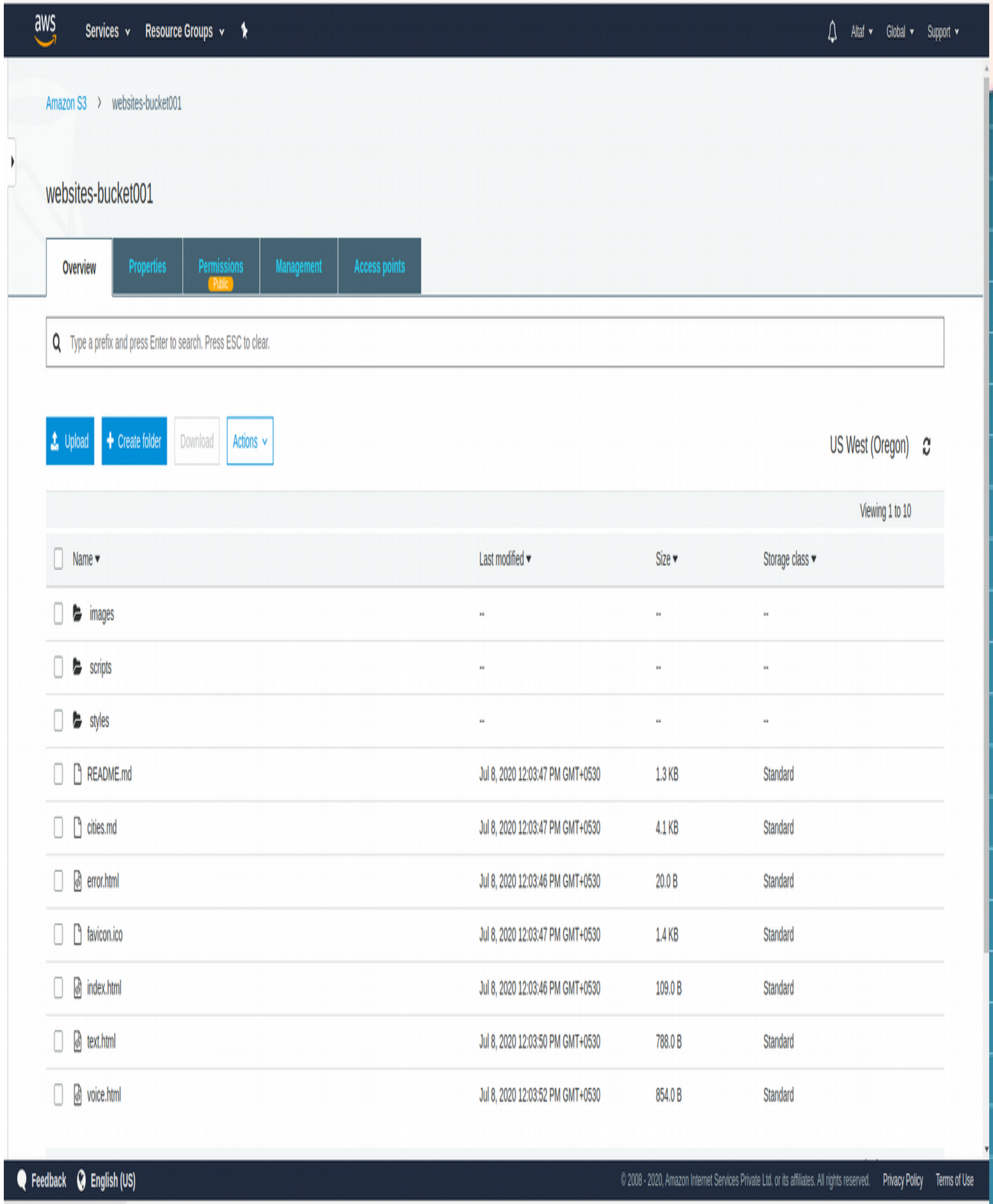
Confirmation prompt

Fulfillment

● AWS Lambda function ○ Return parameters to client

Step2: Create a S3 Bucket and Configure it as Static Website.

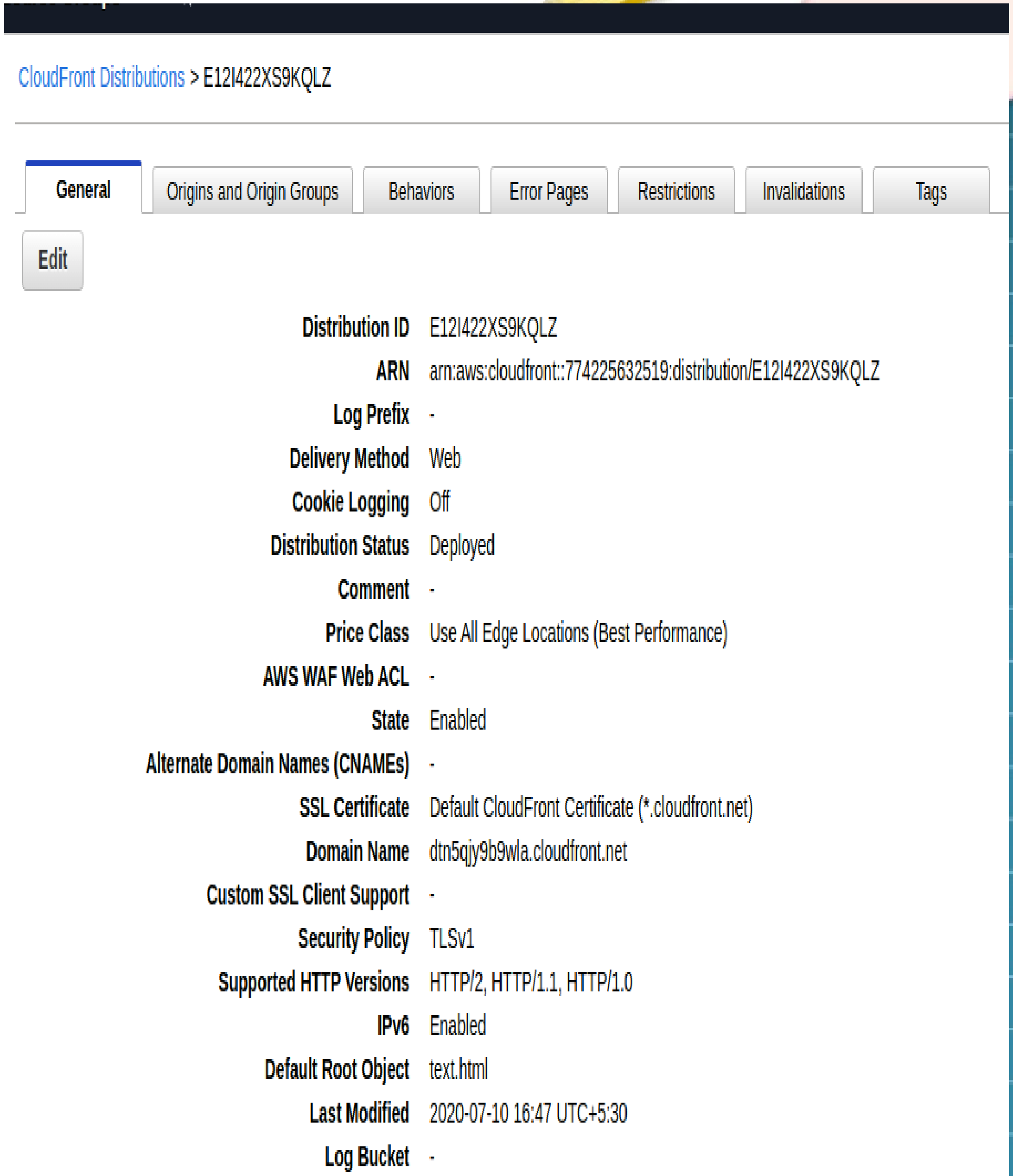
- Create S3 Bucket
- Put your static website into it.
- Put the index named as the starter of the website.
- Please make sure that in every process the region is set to same place, in mine case I have to us-west-2.



Name ▾	Last modified ▾	Size ▾	Storage class ▾
images	--	--	--
scripts	--	--	--
styles	--	--	--
README.md	Jul 8, 2020 12:03:47 PM GMT+0530	1.3 KB	Standard
cities.md	Jul 8, 2020 12:03:47 PM GMT+0530	4.1 KB	Standard
error.html	Jul 8, 2020 12:03:46 PM GMT+0530	20.0 B	Standard
favicon.ico	Jul 8, 2020 12:03:47 PM GMT+0530	1.4 KB	Standard
index.html	Jul 8, 2020 12:03:46 PM GMT+0530	109.0 B	Standard
text.html	Jul 8, 2020 12:03:50 PM GMT+0530	788.0 B	Standard
voice.html	Jul 8, 2020 12:03:52 PM GMT+0530	854.0 B	Standard

Step3: Creating a CloudFront Distribution

- Create Distribution by pointing Origin Domain name to your bucket where you have your website.
- Put Default Root Object to text.html
- Put viewer protocol policy to Redirect Http to Https
- Restrict the S3 bucket policy to CloudFront
- Copy the Url under Domain name present at the CloudFront page.
- And most important first you have to verify your account.

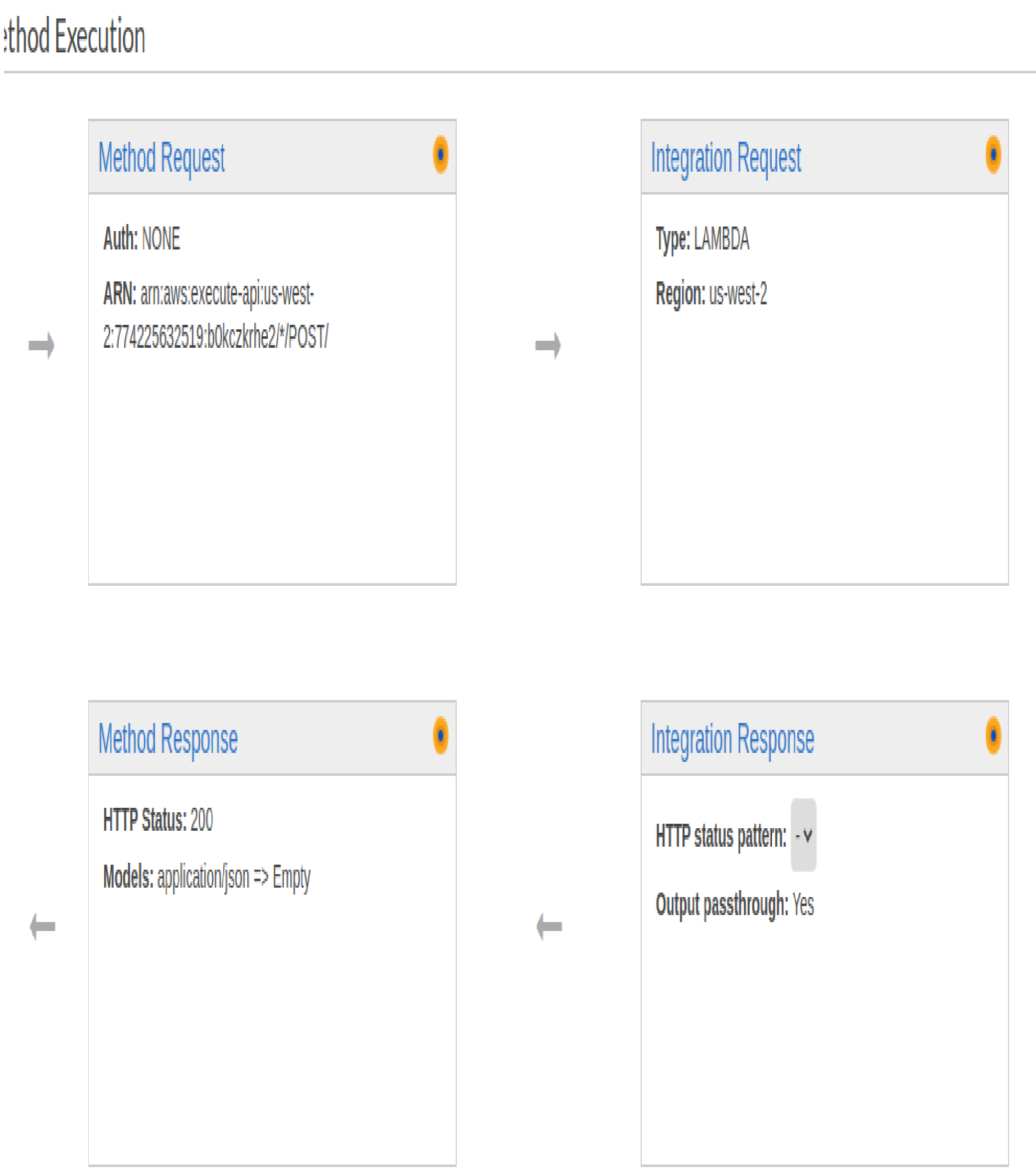


CloudFront Distributions > E12I422XS9KQLZ

Distribution ID	E12I422XS9KQLZ
ARN	arn:aws:cloudfront::774225632519:distribution/E12I422XS9KQLZ
Log Prefix	-
Delivery Method	Web
Cookie Logging	Off
Distribution Status	Deployed
Comment	-
Price Class	Use All Edge Locations (Best Performance)
AWS WAF Web ACL	-
State	Enabled
Alternate Domain Names (CNAMEs)	-
SSL Certificate	Default CloudFront Certificate (*.cloudfront.net)
Domain Name	dtn5qjy9b9wla.cloudfront.net
Custom SSL Client Support	-
Security Policy	TLSv1
Supported HTTP Versions	HTTP/2, HTTP/1.1, HTTP/1.0
IPv6	Enabled
Default Root Object	text.html
Last Modified	2020-07-10 16:47 UTC+5:30
Log Bucket	-

Step4: Create a API Gateway endpoint and connecting to the website.

- Choose REST API and create new API
- Put Actions on Create Method
- For Integration type choose Mock
- On Integration Response under general template choose Method Request passthrough
- Enable CORS and select Default 4XX and Default 5XX
- Deploy API as New Stage
- Copy the invoke URL
- Paste the URL in Config file of the website under API_Gateway_url variable



thod Execution

Method Request

Auth: NONE

ARN: arn:aws:execute-api:us-west-2:774225632519:b0kczkrhe2/*/POST/

Integration Request

Type: LAMBDA

Region: us-west-2

Method Response

HTTP Status: 200

Models: application/json => Empty

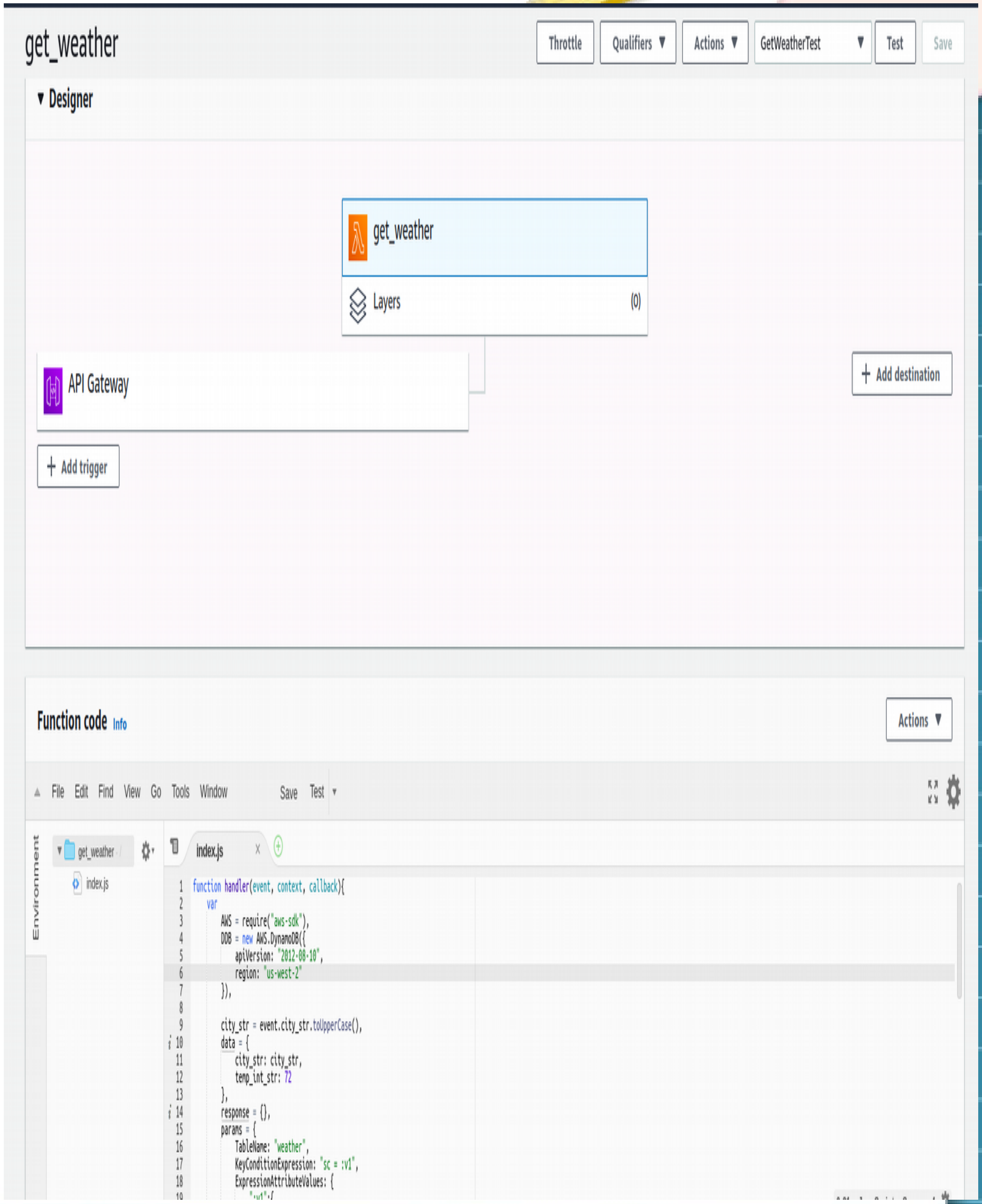
Integration Response

HTTP status pattern: -

Output passthrough: Yes

Step5: Create a Lambda Function

- Create Function by selecting Author from Scratch
- Choose create a new role from AWS policy tempelates
- Run on different test cases
- Replace the API mock endpoint with the current lambda function
- Enable CORS with default 4XX and default 5XX
- Deploy API

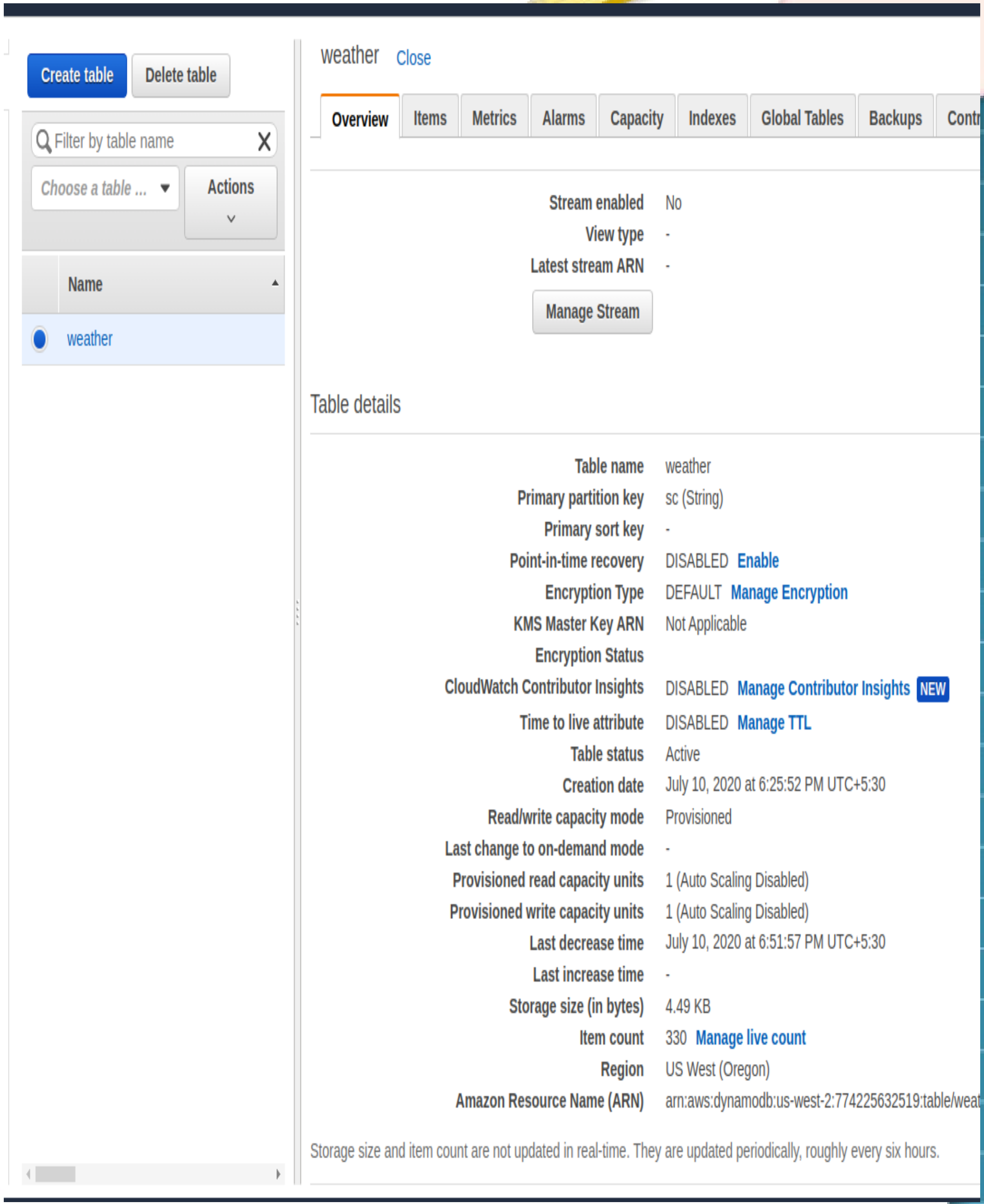


```
function handler(event, context, callback){
    var
        AWS = require("aws-sdk"),
        DDB = new AWS.DynamoDB({
            apiVersion: "2012-08-10",
            region: "us-west-2"
        }),

        city_str = event.city_str.toUpperCase(),
        data = {
            city_str: city_str,
            temp_int_str: 72
        },
        response = {},
        params = {
            TableName: "weather",
            KeyConditionExpression: "sc = :v1",
            ExpressionAttributeValues: {
                ":v1":{
```

Step6: Create a DynamoDB table

- Create Table by giving primary key as ‘sc’
- Remove Use Default Settings
- Put read capacity to 1 and write to 1
- Go to lambda and create function named as seedDynamo
- At Edit Policy add additional permissions and choose a service DynamoDB
- Under Resources choose all resources
- Right Click on the seedDynamo and click new File on the Enviroments tab.
- Name the file cities.csv and paste the city data



Stream enabled No
View type -
Latest stream ARN -

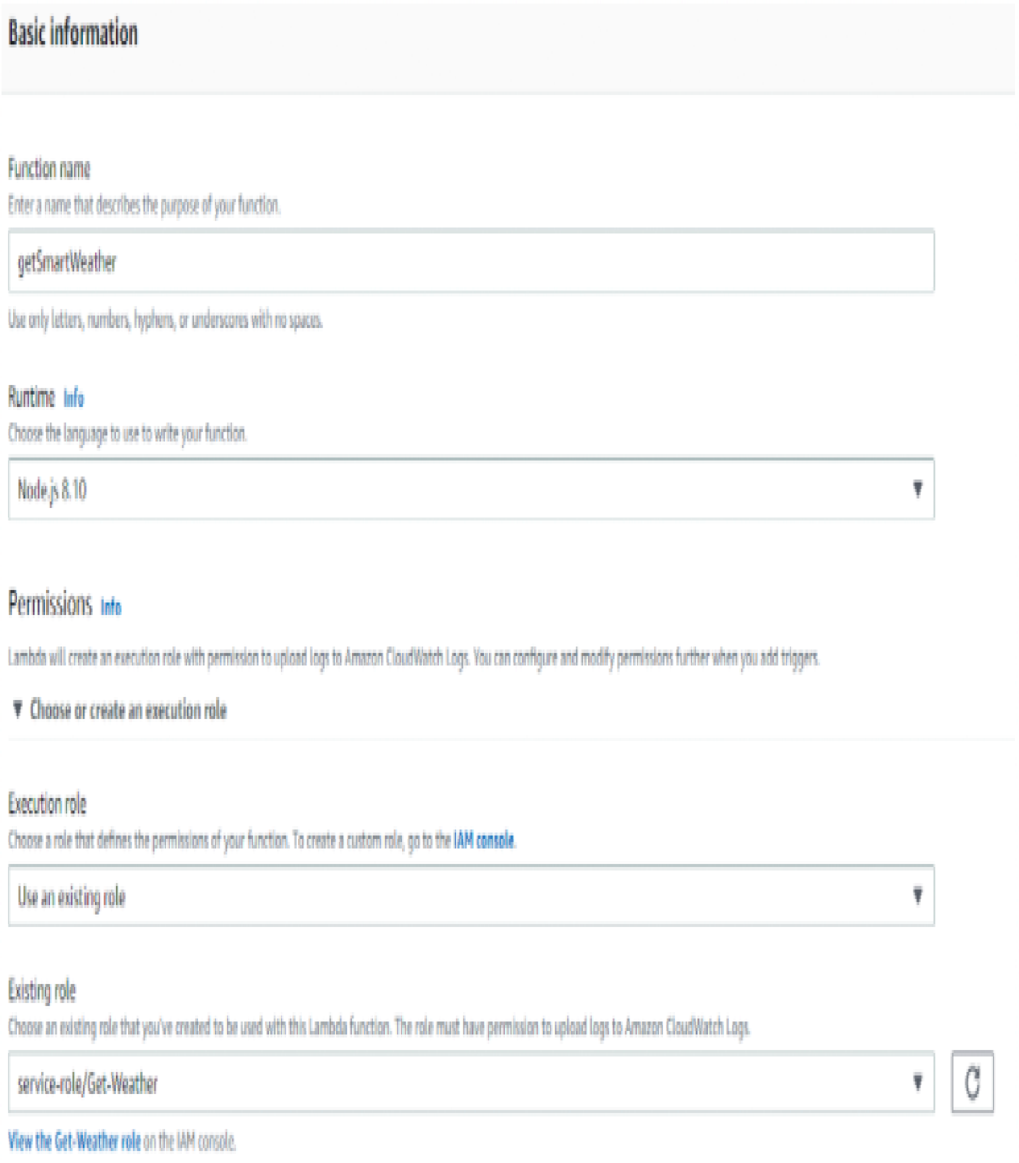
Table details

Table name weather
Primary partition key sc (String)
Primary sort key -
Point-in-time recovery DISABLED Enable
Encryption Type DEFAULT Manage Encryption
KMS Master Key ARN Not Applicable
Encryption Status
CloudWatch Contributor Insights DISABLED Manage Contributor Insights NEW
Time to live attribute DISABLED Manage TTL
Table status Active
Creation date July 10, 2020 at 6:25:52 PM UTC+5:30
Read/write capacity mode Provisioned
Last change to on-demand mode -
Provisioned read capacity units 1 (Auto Scaling Disabled)
Provisioned write capacity units 1 (Auto Scaling Disabled)
Last decrease time July 10, 2020 at 6:51:57 PM UTC+5:30
Last increase time -
Storage size (in bytes) 4.49 KB
Item count 330 Manage live count
Region US West (Oregon)
Amazon Resource Name (ARN) arn:aws:dynamodb:us-west-2:774225632519:table/weat

Storage size and item count are not updated in real-time. They are updated periodically, roughly every six hours.

Step7: Make Lex Smarter

- Create Lambda function getSmartWeather and use an existing role
- Change the timeout to 1:05 min
- Change the index.js code accordingly
- For event name type in getSmartWeatherTest
- Wire it to Lex
- Open our WeatherCatBot in Lex
- Select initialization and validation code hook
- Remove check from Confirmation prompt
- Save intent



Basic information

Function name

Enter a name that describes the purpose of your function.

getSmartWeather

Use only letters, numbers, hyphens, or underscores with no spaces.

Runtime Info

Choose the language to use to write your function.

Node.js 8.10

Permissions Info

Lambda will create an execution role with permission to upload logs to Amazon CloudWatch Logs. You can configure and modify permissions further when you add triggers.

▼ Choose or create an execution role

Execution role

Choose a role that defines the permissions of your function. To create a custom role, go to the IAM console.

Use an existing role

Existing role

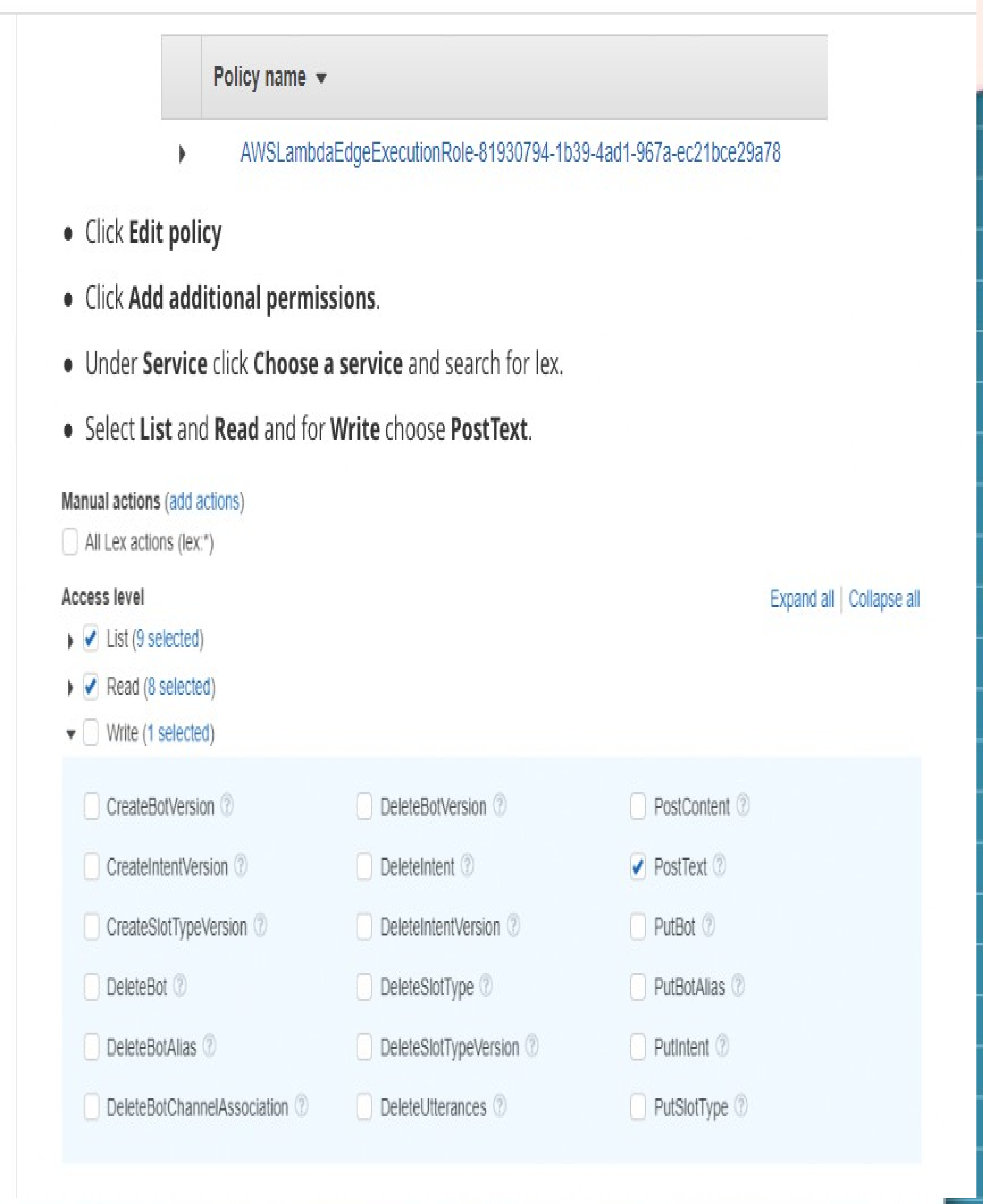
Choose an existing role that you've created to be used with this Lambda function. The role must have permission to upload logs to Amazon CloudWatch Logs.

service-role/Get-Weather

View the Get-Weather role on the IAM console.

Step8:

- Select Lambda and create function and make sure Author from scratch
- Select use an existing role
- For existing role select service-role/Get-Weather
- From policy name click add additional policy
- Under services choose Lex
- From resources select all resources
- Change index.js accordingly



Policy name ▾

AWSLambdaEdgeExecutionRole-81930794-1b39-4ad1-967a-ec21bce29a78

- Click **Edit policy**
- Click **Add additional permissions**.
- Under **Service** click **Choose a service** and search for lex.
- Select **List** and **Read** and for **Write** choose **PostText**.

Manual actions (add actions)

□ All Lex actions (lex:*)

Access level

Expand all | Collapse all

☑ List (9 selected)

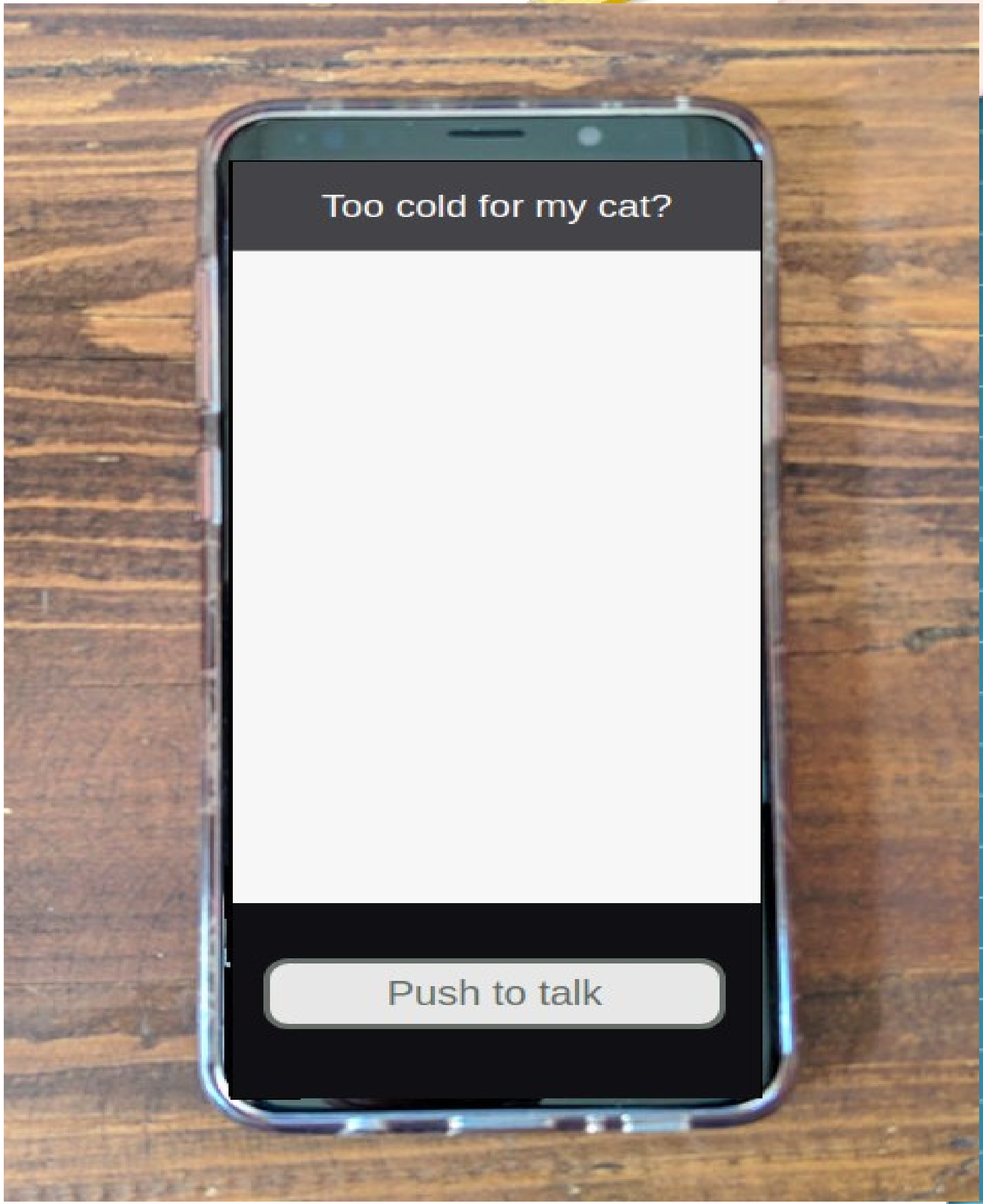
☑ Read (8 selected)

□ Write (1 selected)

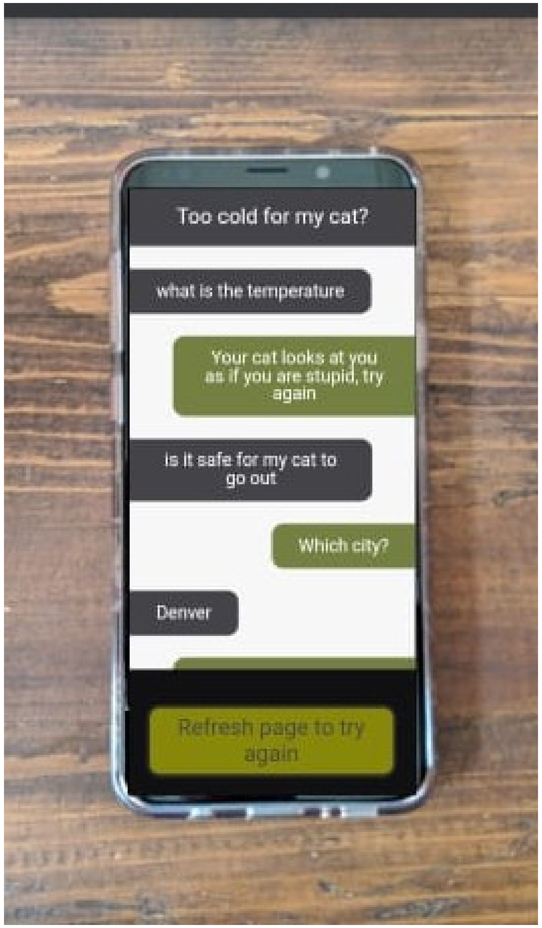
□ CreateBotVersion
□ CreateIntentVersion
□ CreateSlotTypeVersion
□ DeleteBot
□ DeleteBotAlias
□ DeleteBotChannelAssociation
□ DeleteBotVersion
□ DeleteIntent
□ DeleteIntentVersion
□ DeleteSlotType
□ DeleteSlotTypeVersion
□ DeleteUtterances
□ PostContent
☑ PostText
□ PutBot
□ PutBotAlias
□ PutIntent
□ PutSlotType

Step9: Wire Up API Gateway to point to our lex_proxy function

- Go to API Gateway click CatWeather
- Click Integration Request under POST
- ON get_weather type lex_proxy
- Add permission to lambda function
- On actions enable cors and replace existing values
- Deploy the API
- Pull up the CloudFront URL appending */voice.html* at the end
- And It’s Done.



The Final Output



Too cold for my cat?

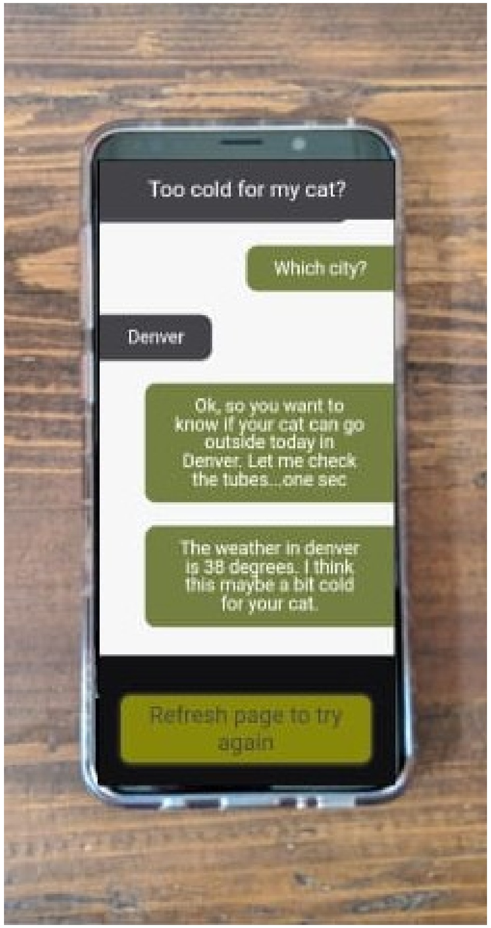
what is the temperature

Your cat looks at you as if you are stupid, try again

is it safe for my cat to go out

Which city?

Denver



Too cold for my cat?

Which city?

Denver

Ok, so you want to know if your cat can go outside today in Denver. Let me check the tubes...one sec

The weather in denver is 38 degrees. I think this maybe a bit cold for your cat.

Conclusion

The Project is being made with the help of the course I did on the Coursera named as **Building Serverless Applications** while completing **AWS fundamentals Specializations**.
It’s a huge achievement for me to complete this specialization and to get to practice many great tools.
This project is just the beginning of a great journey with **AWS** I hope I’ll able to deep dive using this tool and would able to achieve something great from it.

This work is being done by Mohd Altaf