Exercise 4.1: Make Amazon Lex Smarter

Exercise 4.1: Make LEX smarter

We have done well. We have a working server-less data driven website on a content delivery network :)

We now are going to leverage the Amazon LEX bot you created in Week 1 and turn our application from a text app to a voice enabled app.

Therefore the API endpoint that we used for the text application will no longer work if we point it to a Lambda function that is expecting a conversation LEX-style "phrase".

So, out with the old and in with the new.

The goal of this exercise is to swap out the Lambda function currently sitting behind API Gateway for a new one that simply takes a "text phrase" from the browser and passes it over to LEX to interpret. When LEX replies to this message we pass that messages all the way back to the browser.

This new lambda function's code is just passing messages back and forth between the browser and LEX, nothing more. We will call this function `lexproxy`

So before I explain the other steps, you might be thinking. How can a browser send text to API Gateway through Lambda to LEX and back, if I am speaking not typing?

Well, we are going to cheat ;)

We are leveraging the fact that the Chrome browser can listen to your device microphone and convert what you say to text in real time. This is a JavaScript API, that you don't need to worry about as we wrote the code for this already. There is another JavaScript API that converts text back to speech! Super useful. We shall be leveraging that too :)

This way we can literally "talk" to the browser, and essentially be communicating with LEX, in what feels like "real time".

We are going to approach this exercise in multiple steps:

1) We need to make your Lex bot smarter with what we call a **validation hook**. This way it can extract data from DynamoDB and provide weather information. Just like our text app did.

2) Test you smarter LEX bot in the LEX console. You should notice the difference in smarts

2) Create a lambda function to proxy text phrases to and from your new smarter LEX bot, and text it.

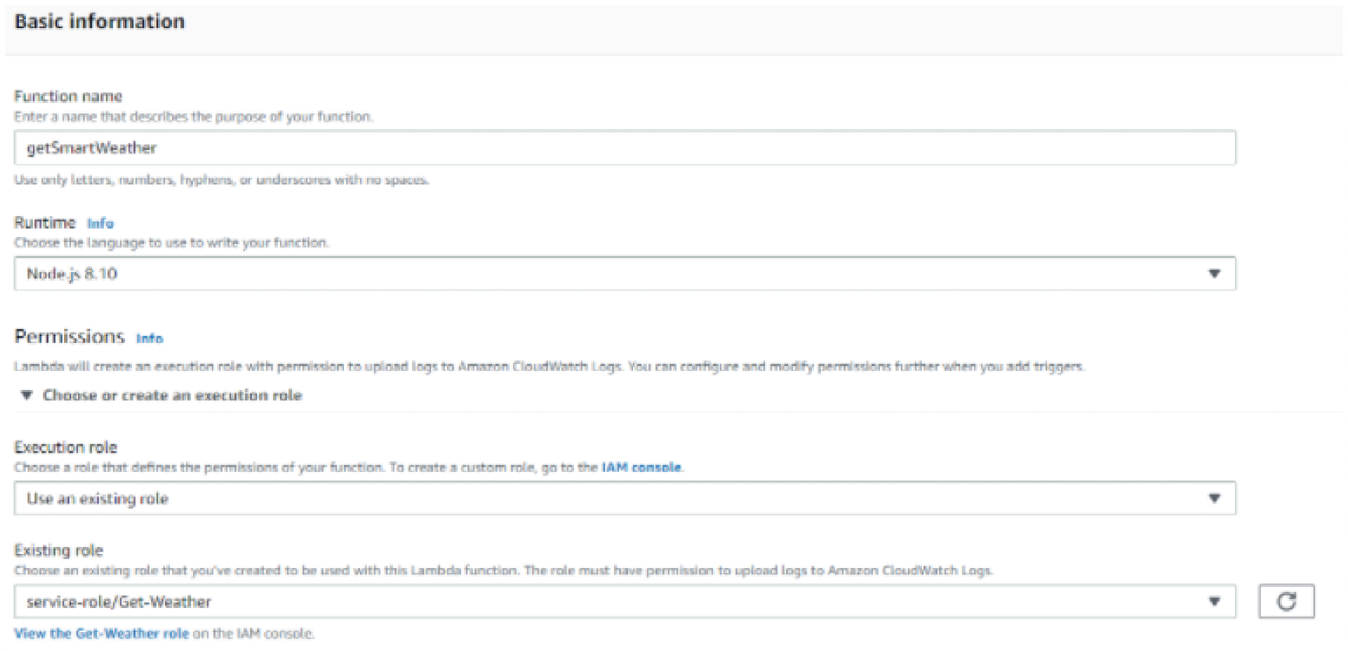
3) Change the API gateway configuration so it points to this new proxy function, and test it.

4) Go to a NEW secret website (that you upload when you did week 2) and may not have realized it called `voice.html`.

1. Steps to creating a validation Lambda function for the Lex bot.

This will make LEX smarter :)

- Sign in to the AWS Management Console and in the **Find Services** search box type lambda and choose **Lambda**.
- Click **Create function**.
- Select **Author from scratch**.
- For **Function name** type in `getSmartWeather`.
- Again leave the **Runtime** as **Node.js 8.10**.
- For **Execution role** leave it as **Use an existing role**.
- For **Existing role** choose **service-role/Get-Weather**.



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.

- Click **Create function**.
- Paste the following into the `index.js` tab in the inline code editor.

```
exports.handler = function(event, ctx, cb){
  var
    my_response = {};
    if(event.currentIntent.slots.city_str){
        // we have the city already awesome keep going
    }else{
        //we need to ask for (elicit) a city
        my_response.statusCode = 200;
        my_response.body = {
            "dialogAction": {
                "type": "ElicitSlot",
                "message": {
                    "contentType": "PlainText",
                    "content": "Name the city your cat lives in, thanks"
                },
                "intentName": "CatWeather",
                "slots": {
                    "city_str": null
                },
                "slotToElicit" : "city_str"
            }
        };
        return cb(null, my_response.body);
    }
    var
        city_str = event.currentIntent.slots.city_str,
        AWS = require("aws-sdk"),
        DDB = new AWS.DynamoDB({
            apiVersion: "2012-08-10",
            region: "us-east-1"
        }),
        lookup_name_str = city_str.toUpperCase(),
        params = {
            TableName: "weather",
            KeyConditionExpression: "sc = :v1",
            ExpressionAttributeValues: {
                ":v1":{
                    "S": lookup_name_str
                }
            },
            ProjectionExpression: "t"
        };

    console.log(params);
    DDB.query(params, function(err, data){
        if(err){
            throw err;
        }

        if(data.Items && data.Items[0] && data.Items[0].t){
            console.log("city weather found");
            console.log(data.Items[0]);
            my_response.statusCode = 200;
            my_response.body = {
                "sessionAttributes": {
                    "temp_str": data.Items[0].t.N,
                    "city_str": event.currentIntent.slots.city_str
                },
                "dialogAction":{
                    "type": "Close",
                    "fulfillmentState": "Fulfilled",
                    "message": {
                        "contentType": "PlainText",
                        "content": data.Items[0].t.N
                    }
                }
            };
        }else{
            console.log("city weather not found for " + lookup_name_str);
            my_response.statusCode = 200;
            my_response.body = {
                "dialogAction": {
                    "type": "ElicitSlot",
                    "message": {
                        "contentType": "PlainText",
                        "content": "Please try another city, we couldn't find
                            the weather for that city"
                    },
                    "intentName": "CatWeather",
                    "slots": {
                        "city_str": null
                    },
                    "slotToElicit" : "city_str"
                }
            }
        }
        return cb(null, my_response.body);
    });
};
```

This code checks for an existence of a slot (the city), if it's there, wonderful. Look it up in Dynamo and return the temperature.

If not, LEX will need to ask for a city. Once LEX has established the city, meaning there is existence of the slot, it can just do a look up.

You get the idea.

- Scroll down to **Basic settings** and change the timeout to **1** min and **5** sec.
- Click **Save**.
- Let's test it with a dummy LEX payload before we tie it into Lex and publish it. Click **Test**.
- Leave the **Event template** as **Hello World**.
- For **Event name** type in `getSmartWeatherTest`.
- Paste the following into the text box:

```
{
  "messageVersion": "1.0",
  "invocationSource": "DialogCodeHook",
  "userId": "1012602",
  "sessionAttributes": {
  },
  "bot": {
    "name": "WeatherCatBot",
    "alias": "$LATEST",
    "version": "$LATEST"
  },
  "outputDialogMode": "Text",
  "currentIntent": {
    "name": "CatWeather",
    "slots": {
      "city_str": "CHICAGO"
    },
    "confirmationStatus": "None"
  }
}
```

- Click **Create**.
- Click **Test**.
- We should see the following output:

```
{
  "sessionAttributes": {
    "temp_str": "42",
    "city_str": "CHICAGO"
  },
  "dialogAction": {
    "type": "Close",
    "fulfillmentState": "Fulfilled",
    "message": {
      "contentType": "PlainText",
      "content": "42"
    }
  }
}
```

- Lets edit our test by clicking getSmartWeatherTest ▼ and choosing **Configure test events**.
- Let's add **1** to our `city_id` which will be `1012603` (this effectively gives you a new session) and change the `city_str` to `BANANA`. < my favorite city ;)

```
{
  "messageVersion": "1.0",
  "invocationSource": "DialogCodeHook",
  "userId": "1012603",
  "sessionAttributes": {
  },
  "bot": {
    "name": "weather",
    "alias": "$LATEST",
    "version": "$LATEST"
  },
  "outputDialogMode": "Text",
  "currentIntent": {
    "name": "catWeather",
    "slots": {
      "city_str": "BANANA"
    },
    "confirmationStatus": "None"
  }
}
```

- Click **Save**.
- Click **Test**.
- You should see the following output:

```
{
  "dialogAction": {
    "type": "ElicitSlot",
    "message": {
      "contentType": "PlainText",
      "content": "Please try another city, we couldn't find the weather for
          that city"
    },
    "intentName": "CatWeather",
    "slots": {
      "city_name_str": null
    },
    "slotToElicit": "city_str"
  }
}
```

- Lets make one more edit to our test. Click getSmartWeatherTest ▼ and **Configure test events**.
- Once again increment the `city_id` to `1012604` and set the `city_str` to `null`. As if we didn't pass in a city in our phrase. i.e "is it cold out?"

```
{
  "messageVersion": "1.0",
  "invocationSource": "DialogCodeHook",
  "userId": "1012604",
  "sessionAttributes": {
  },
  "bot": {
    "name": "weather",
    "alias": "$LATEST",
    "version": "$LATEST"
  },
  "outputDialogMode": "Text",
  "currentIntent": {
    "name": "catWeather",
    "slots": {
      "city_str":  null
    },
    "confirmationStatus": "None"
  }
}
```

- Click **Save**.
- Click **Test**.
- You should see the following output:

```
{
  "dialogAction": {
    "type": "ElicitSlot",
    "message": {
      "contentType": "PlainText",
      "content": "Name the city your cat lives in, thanks"
    },
    "intentName": "CatWeather",
    "slots": {
      "city_str": null
    },
    "slotToElicit": "city_str"
  }
}
```

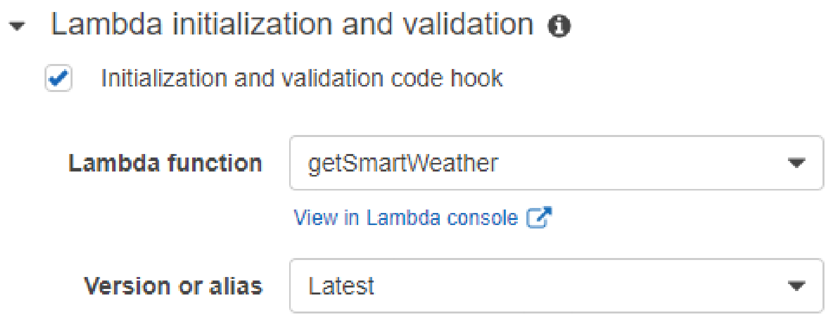
Awesome it is all working, we now have a function that we know works

We just need to wire it up to LEX and try it in the LEX console.

Lets give LEX it's new brain :)

2. Wire it into Lex - (Turbo charge LEX)

- Click **Services** type in lex in the search box or select **Amazon Lex** from the **History** list.
- Click our **WeatherCatBot**
- Scroll down to **Lambda initialization and validation** and select **Initialization and validation code hook**.
- Select `getSmartWeather` for our function and **Latest** for **Version or alias**.



Lambda initialization and validation

[✓] Initialization and validation code hook

Lambda function: getSmartWeather

Version or alias: Latest

- At the **Add permission to Lambda Function** click **OK**.
- Expand the **Confirmation prompt** and remove the check for **Confirmation prompt**.
- Click **Save Intent** at the bottom.
- Click **Build** at the top right.
- Click **Build** again at the **Build your bot** pop-up.

3. Test our bot with Lambda

- Click **Test Chatbot** at the right if it is not already expanded.
- Test out talking to your bot.

User 🙍	**Chatbot** 🤖
Can my cat go out in banana?	
	Please try another city, we couldn't find the weather for that city
Can my cat go out in DENVER	
	38
Can my cat go out?	
	Name the city your cat lives in, thanks
alto	
	47
Will my cat be OK outside?	
	Name the city your cat lives in, thanks
Tempe	
	31

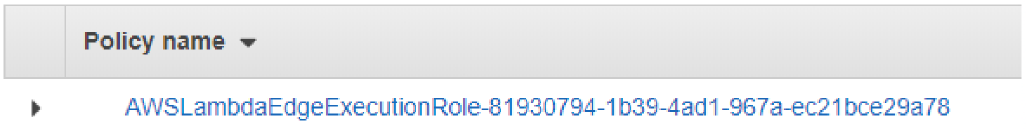
I hope you see now that LEX is much smarter and feels more human. Yes it just spits out a temperature when done, but that's ok, because we already have front end JavaScript in place that provides an opinion on if your cat should go out or not if you provide it a temp. So we are all set really.

We just need to create the new Lambda proxy now and swap it out in API gateway. This will disable the old text based app and switch us to the realm of voice!

..we will do service tests as we go.

4. Steps for creating the lambda messaging proxy.

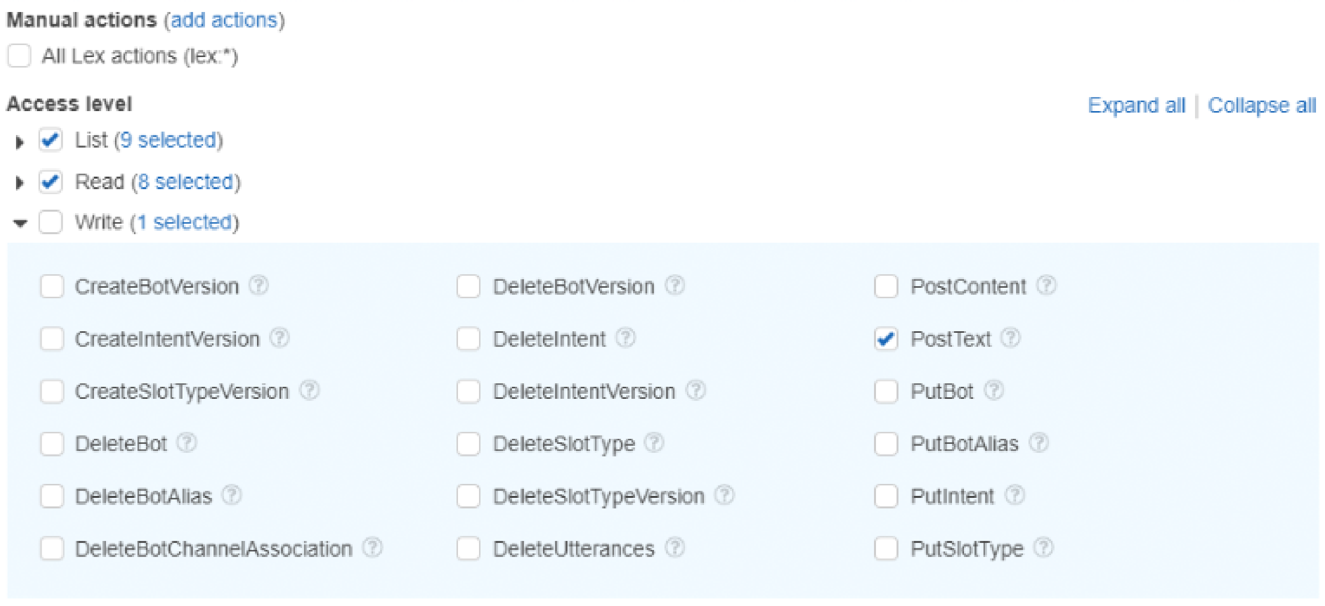
- Click **Services** type in lambda in the search box or select **Lambda** from the **History** list.
- Click **Create function**.
- Make sure **Author from scratch** is selected.
- For **Function name** type in `lex_proxy`.
- For **Runtime** leave it as **Node.js 8.10**.
- For **Execution role** select **Use an existing role**.
- For **Existing role** select **service-role/Get-Weather**.
- Click **View the Get-Weather role** on the IAM console.
- Click the drop-down arrow next to the **Policy name**.



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

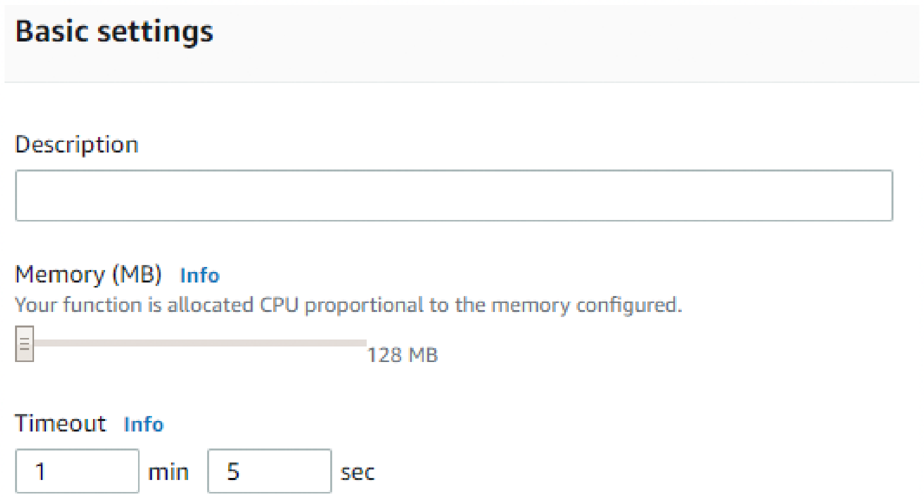
[✓] List (9 selected)

[✓] Read (8 selected)

[ ] Write (1 selected)

[ ] CreateBotVersion	[ ] DeleteBotVersion	[ ] PostContent
[ ] CreateIntentVersion	[ ] DeleteIntent	[✓] PostText
[ ] CreateSlotTypeVersion	[ ] DeleteIntentVersion	[ ] PutBot
[ ] DeleteBot	[ ] DeleteSlotType	[ ] PutBotAlias
[ ] DeleteBotAlias	[ ] DeleteSlotTypeVersion	[ ] PutIntent
[ ] DeleteBotChannelAssociation	[ ] DeleteUtterances	[ ] PutSlotType

- Under **Resources** select **All resources**.
- Click **Review policy**.
- Click **Save changes**.
- Go back to your **Lambda** tab.
- Click **Create function**.
- Scroll down to **Basic settings** and set the **Timeout** to **1** min and **5** sec.


Basic settings

Description

Memory (MB) Info
Your function is allocated CPU proportional to the memory configured.
128 MB

Timeout Info
1 min 5 sec

- Paste the following into the `index.js` tab:

```
function handler(event, context, callback){
    var
        MESSAGE_STR = event.message_str,
        USER_ID_STR = event.user_id_str,
        AWS = require("aws-sdk"),
        LEXRUNTIME = {},
        BOT_NAME_STR = "WeatherCatBot",
        BOT_ALIAS_STR = "$LATEST",
        sessionAttributes = {

        },
        params = {};

    AWS.config.update({
        region: "us-east-1"
    });

    LEXRUNTIME = new AWS.LexRuntime();

    params = {
        botAlias: BOT_ALIAS_STR,
        botName: BOT_NAME_STR,
        inputText: MESSAGE_STR,
        userId: USER_ID_STR,
        sessionAttributes: sessionAttributes
    };
    LEXRUNTIME.postText(params, function(error, data){
        var response = {};
        if(error){
            console.log(error, error.stack);
            response = "problem with lex";
            callback(null, response);
        }else{
            console.log(data);
            response = data;
            callback(null, response);
        }
    });
}
exports.handler = handler;
```

This code sends what you give it to LEX, and sends the reply from LEX back out. i.e A proxy.

- Click **Save**.

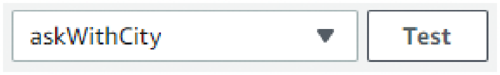
5. Steps for testing it in the Lambda console

- Still in our Lambda console. Click **Configure Test**.
- Leave **Event template** as **Hello World**.
- For **Event name** type in `askWithCity`.
- Paste the following code: *(the user_id_str is to maintain the session for that user)*

```
{
  "message_str": "can my cat go out in alto?",
  "user_id_str": "10126023"
}
```

- Click **Create**.
- Click **Test**.
- You should see a similar response: *Perfect!*

```
Response:
{
  "intentName": "CatWeather",
  "slots": {
    "city_str": "alto"
  },
  "sessionAttributes": {
    "city_str": "alto",
    "temp_str": "47"
  },
  "message": "47",
  "messageFormat": "PlainText",
  "dialogState": "Fulfilled",
  "slotToElicit": null
}
```

- We can try it without a city.
- Click the drop-down arrow next to 

- Click **Configure test events**.
- Change the event: (new user id - bump)

```
{
  "message_str": "can my cat go out in alto?",
  "user_id_str": "10126023"
}
```

To

```
{
  "message_str": "can my cat go out?",
  "user_id_str": "10126024"
}
```

- Click **Save**.
- Click **Test**.
- You should see the following output:

```
{
  "intentName": "CatWeather",
  "slots": {
    "city_str": null
  },
  "sessionAttributes": {},
  "message": "Name the city your cat lives in, thanks",
  "messageFormat": "PlainText",
  "dialogState": "ElicitSlot",
  "slotToElicit": "city_str"
}
```

- Modify the `askWithCity` test case again with the following: This time keep the user id, as you are in mid conversation, and do not want to start a new session.

```
{
  "message_str": "DENVER",
  "user_id_str": "10126024"
}
```

- You should see the following output:

```
{
  "intentName": "CatWeather",
  "slots": {
    "city_str": "DENVER"
  },
  "sessionAttributes": {
    "city_str": "DENVER",
    "temp_str": "38"
  },
  "message": "38",
  "messageFormat": "PlainText",
  "dialogState": "Fulfilled",
  "slotToElicit": null
}
```

- Since the `user_id_str` didn't change it will keep that same session open and return the correct data.

Awesome, you are nearly done.

We now just point API Gateway to this new proxy, disabling the old text (text.html) API. That webpage will no longer work.

Once you follow these next few steps you can visit the `/voice.html` version of the website and test it.

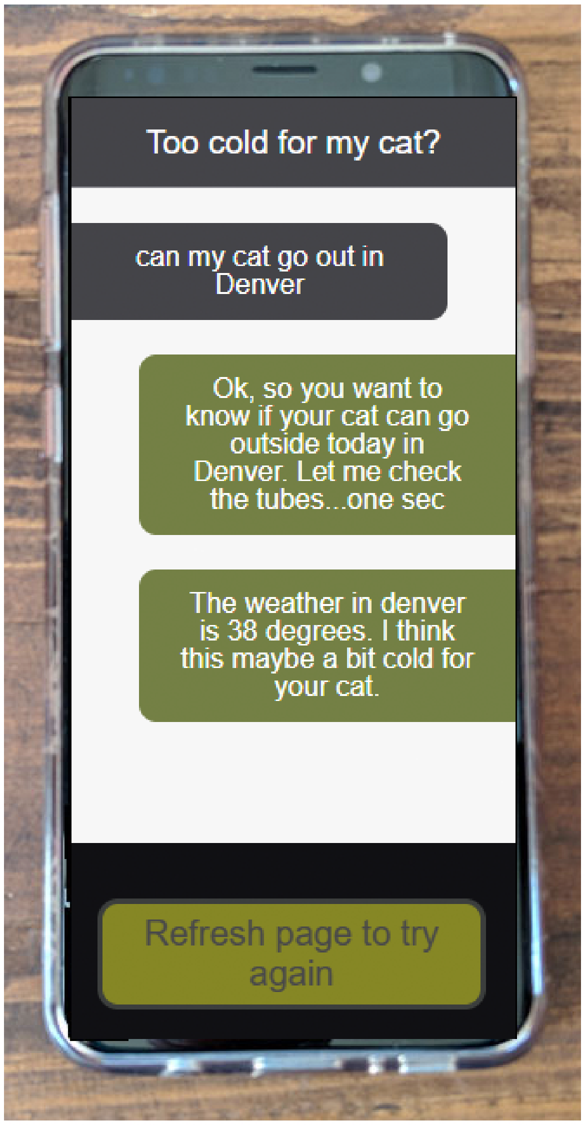
---

6. Final steps: Wire up API Gateway to point to our `lex_proxy` function.

- Sign in to the AWS Management Console and in the **Find Services** search box type api and choose **API Gateway** or choose **API Gateway** from the **History** list.
- Under **APIs** click **CatWeather**.
- Click **POST** POST
- Click **Integration Request**.
- Click the pencil icon next to `get_weather`
- Type in **l** and select `lex_proxy` and click the check ✓ to update it.
- Click **OK** on the **Add Permission to Lambda Function** pop-up.
- Go back to the root ▾ / under **Resources**.
- Click **Actions** and **Enable CORS**. Click **DEFAULT 4XX** and **DEFAULT 5XX**.
- Click **Enable CORS and replace existing CORS headers**.
- Click **Yes, replace existing values** on the **Confirm method changes** pop-up.
- Click **Actions** and **Deploy API**. Choose **test** for the **Deployment stage** and click **Deploy**.
- Pull up your CloudFront URL appending `/voice.html` at the end

Ensure you have your microphone enabled if you see a pop up in the browser. Click **Push to talk** and utter a phrase like: "Can my cat go out in Denver?"

Your bot will chat back with you :)



Too cold for my cat?

can my cat go out in 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.

Congrats you are done.

BEFORE you close this out and head over to the next video, please tear down any applications you no longer want running. Outside of free tier some of these services are not free.**

Check the forums if you need help tearing down the services.

Exercise goal checklist

1. ~~Create a simple chatbot using the lex console.~~
2. ~~Upload our website to S3.~~
3. ~~Create a content delivery network and lock down S3.~~
4. ~~Build an API gateway mock with CORS.~~
5. ~~Build a Lambda mock, use IAM, push logs to CloudWatch.~~
6. ~~Create and seed a database with weather data.~~
7. ~~Enhance the lambda, so it can query the database.~~
8. ~~Play with your new text based data driven application.~~
9. ~~Create a LEX proxy using Lamba.~~
10. ~~Enhance API gateway to use the LEX proxy.~~
11. ~~Play with your new voice web application.~~