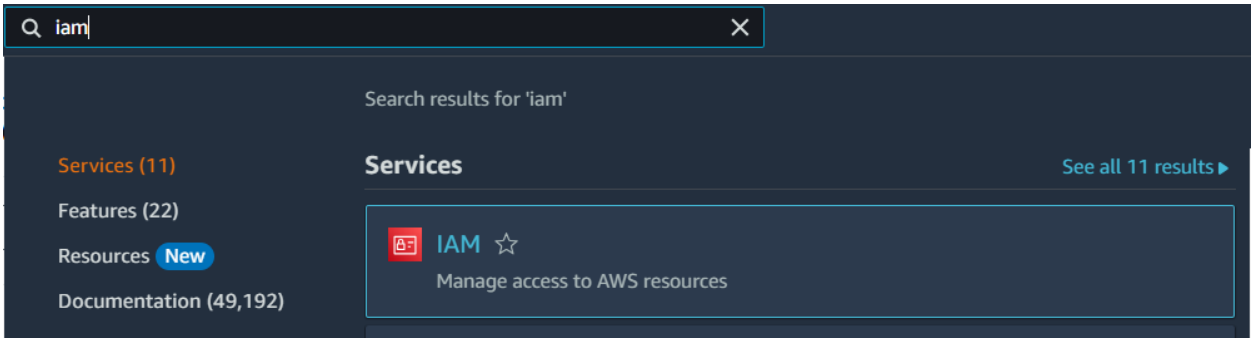
Text narrator using Amazon Polly

Amazon Polly is a service provided by AWS that enables developers to generate human-like speech from text.

- **Text-to-Speech Conversion**: Amazon Polly turns written text into spoken words. So, you can type something, and Amazon Polly will say it out loud in a natural-sounding voice.
- **Realistic Speech**: The speech created by Amazon Polly sounds like a real person speaking, not robotic or unnatural. It's great for making computerized voices sound more human-like.
- **Options to Customize**: You can change how the speech sounds. For example, you can make it faster or slower, adjust the pitch (high or low), and even pick different accents or languages.
- **Supports Many Languages and Voices**: Amazon Polly can speak in lots of different languages and with different voices. Whether you want a man or a woman, or someone with a British or American accent, you have options.
- **SSML Support for Control**: You can use special codes called SSML to control exactly how the speech sounds. This allows for things like emphasizing certain words, adding pauses, or changing the tone of voice.

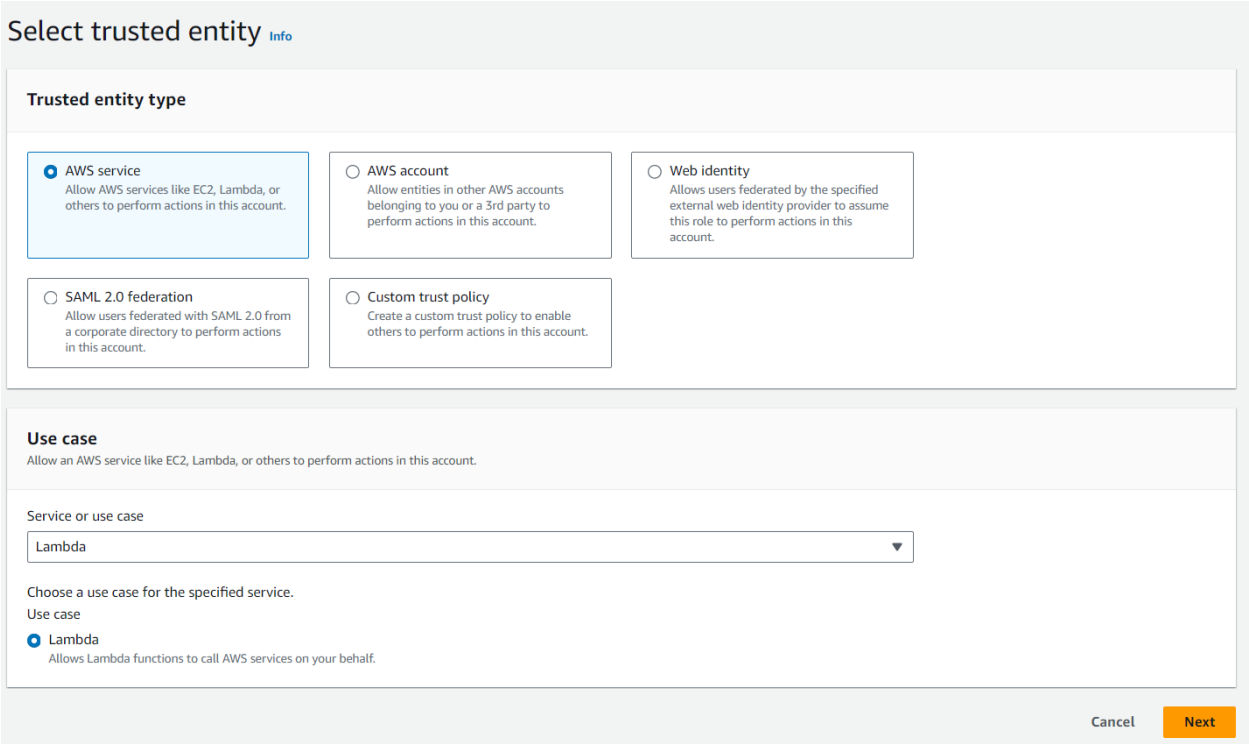
Creating an IAM role

1. In this project, we will try to access the Amazon Polly service and store the audio output in a S3 bucket using a Lambda function.
2. For that we need an **IAM role** with suitable policies attached to it.
3. From your **AWS management console**, search for **IAM** from the search bar.



4. Navigate to **Access Management → Roles → Create Role**.

5. Select AWS service as the trusted entity type and Lambda as your use case. Click on **Next**.



6. From the list of permission policies, choose the following policies :

    1. *AmazonPollyFullAccess*

2. *AmazonS3FullAccess*

3. *AWSLambdaBasicExecutionRole*

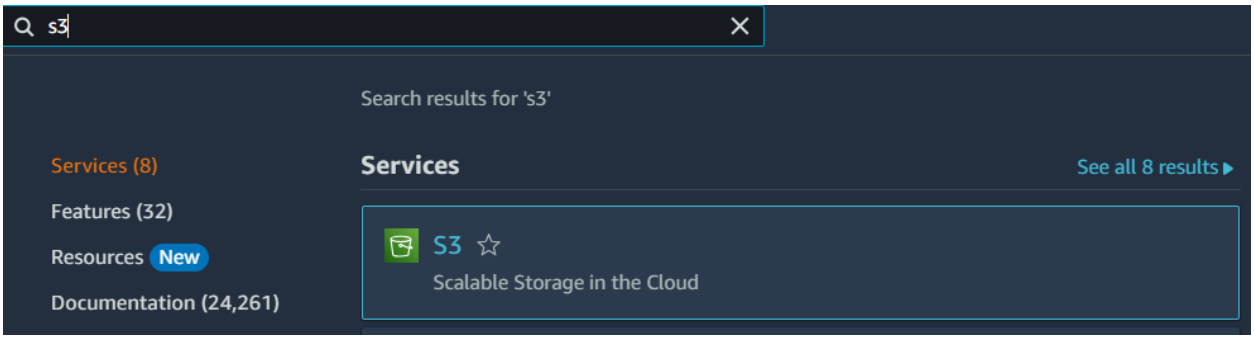
7. Click on **Next**.

8. Provide a suitable name and description for the IAM role and click on **Create role.**

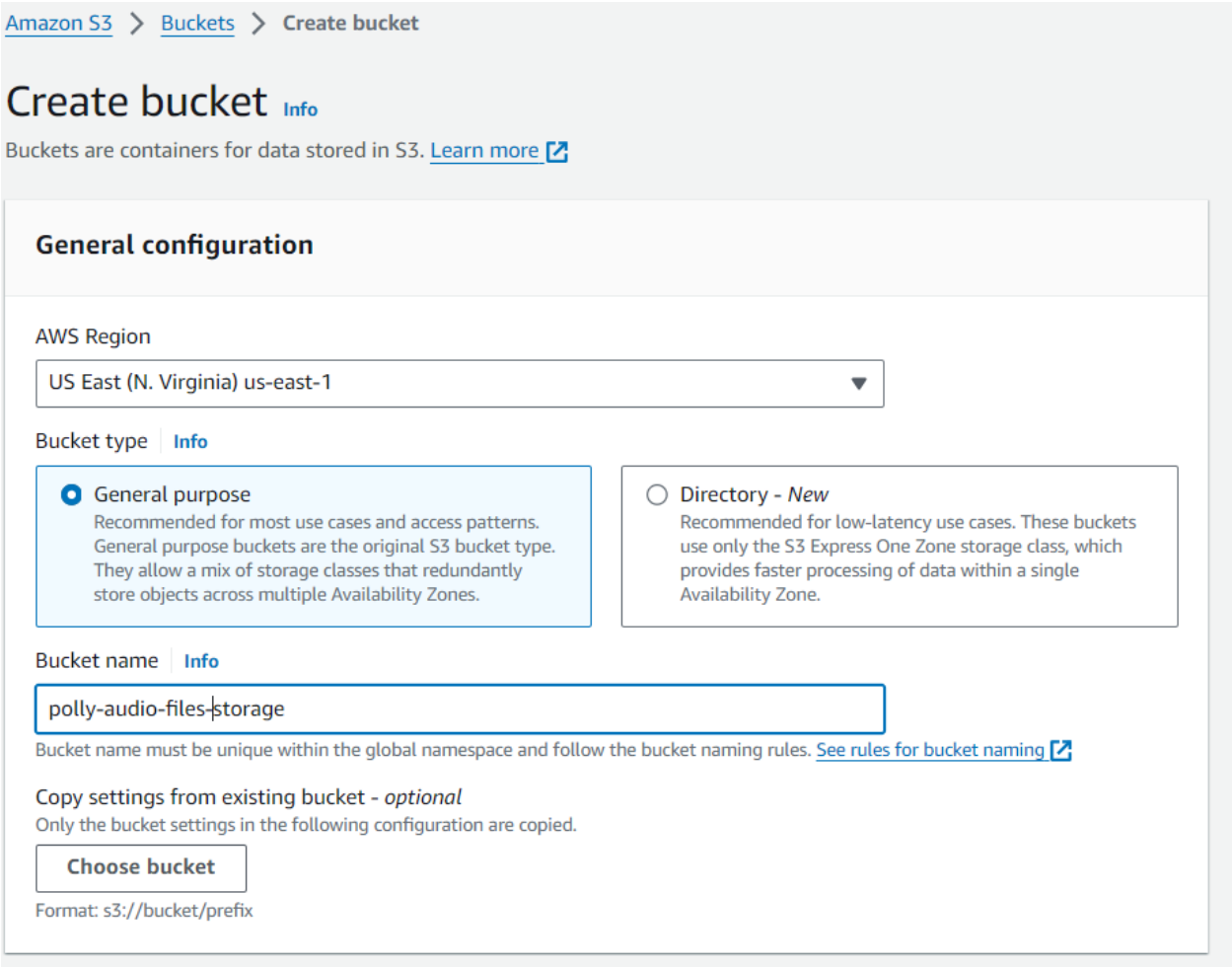
Creating a S3 bucket

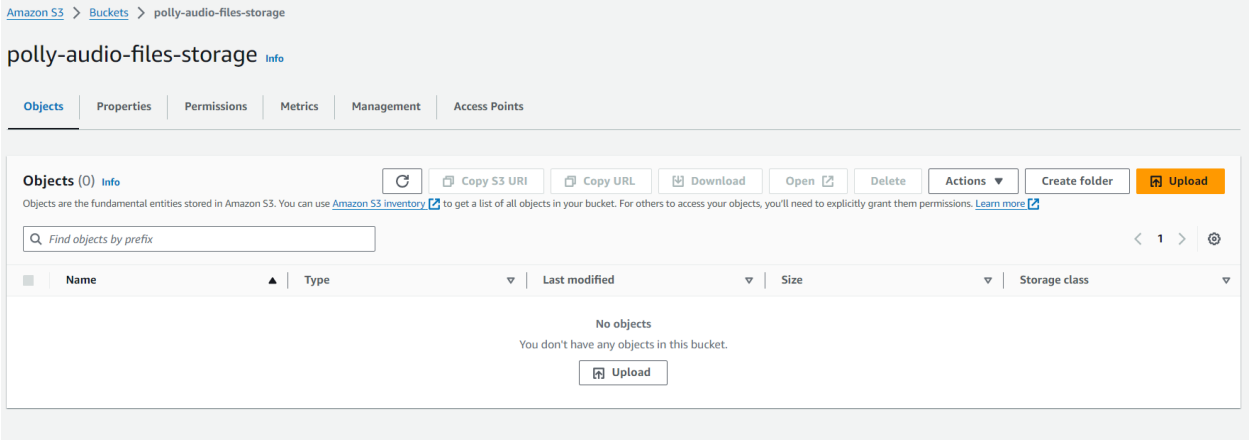
1. From your AWS management console, search for S3 from the search bar.

2. Click on **Create bucket.**



3. Give a suitable name for the S3 bucket, keep the rest of the configurations as it is and click on **Create bucket.**

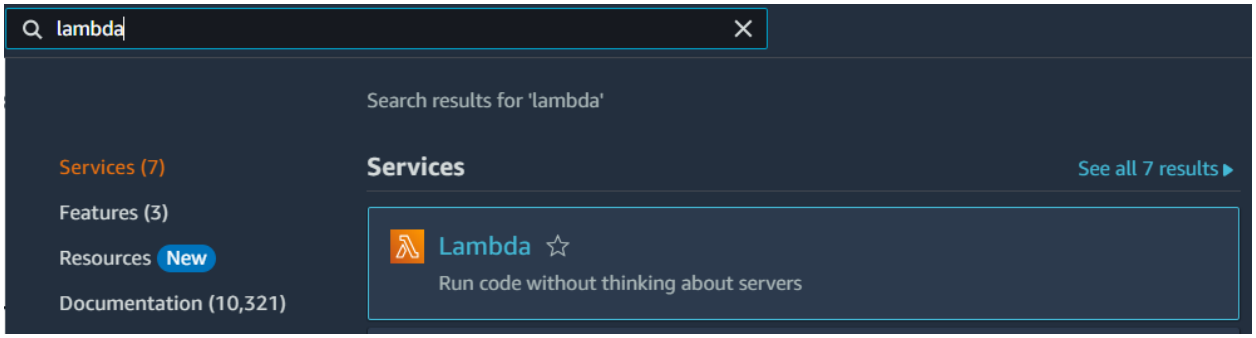




4. The audio files from generated by Amazon Polly would be stored in this bucket with the help of AWS Lambda.

Creating a Lambda function

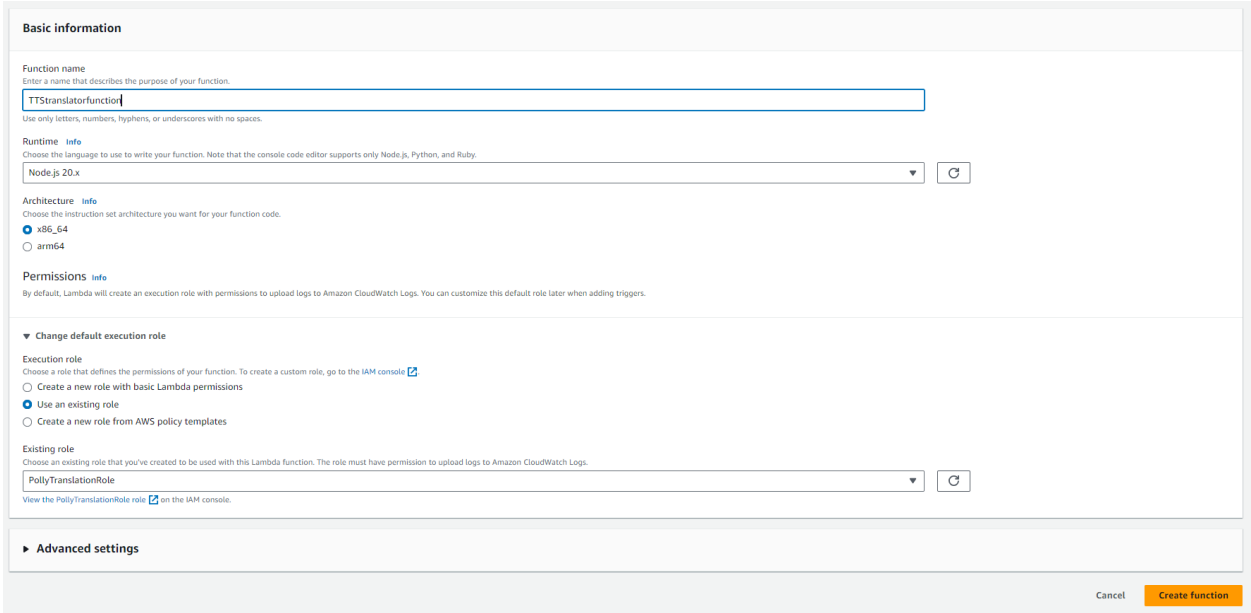
1. From your AWS management console, navigate to Lambda from the search bar.



2. Click on **Create function.**

3. Give an appropriate name to the Lambda function and choose **Node.js 16.x** as the runtime environment.

5. Toggle the option '**Change default configuration role**' and check '**Use existing role**'.
6. Choose the role that you created earlier, leave the rest of configurations as default and click **Next.**



7. Rename your **'index.mjs'** file to **'index.js'.**
8. We're using Amazon's tools (AWS SDK) to talk to two services: **Polly** (for making speech from text) and **S3** (for storing files).

```javascript
const { PollyClient, SynthesizeSpeechCommand } = require("@aws-sdk/client-polly");
const { S3Client } = require("@aws-sdk/client-s3");
const { Upload } = require("@aws-sdk/lib-storage");

const polly = new PollyClient({});
const s3 = new S3Client({});
```

9. We're writing a function that AWS will run for us whenever something happens. It's like a little program that waits for a signal to start working.

```javascript
exports.handler = async (event) => {
```

10. When the function gets a message with some text, we're going to make it into speech. We decide how the speech will sound and what format it should be in

```javascript
const text = event.text;

const params = {
    Text: text,
    OutputFormat: 'mp3',
    VoiceId: 'Joanna' // You can change this to the desired voice
};
```

11. We send the text to Polly and ask it to turn it into speech. Polly does its magic and gives us back the speech as data.
12. We then save this speech in our S3 storage.

```javascript
// Synthesize speech using Polly
        const command = new SynthesizeSpeechCommand(params);
        const data = await polly.send(command);

        // Generate a unique key for the audio file
        const key = `audio-${Date.now()}.mp3`;

        // Use Upload to stream the audio file to S3
        const upload = new Upload({
            client: s3,
            params: {
                Bucket: "<YOUR-BUCKET-NAME>", //Replace with your bucket name
                Key: key,
                Body: data.AudioStream, // Streamed body
                ContentType: "audio/mpeg",
            },
        });

        await upload.done(); // Wait for upload to complete
```

13. We make a message saying the speech has been saved successfully with its special name in our storage. If something goes wrong, there is an error message.

```javascript
return {
            statusCode: 200,
            body: JSON.stringify({ message: `Audio file stored as ${key}` }),
        };
    } catch (error) {
        console.error("Error:", error);
        return {
            statusCode: 500,
            body: JSON.stringify({ message: "Internal server error" }),
        };
    }
};
```

Complete Lambda Function

Your complete Lambda function code will look like this.

```javascript
const { PollyClient, SynthesizeSpeechCommand } = require("@aws-sdk/client-polly");
const { S3Client } = require("@aws-sdk/client-s3");
const { Upload } = require("@aws-sdk/lib-storage");

const polly = new PollyClient({});
const s3 = new S3Client({});

exports.handler = async (event) => {
    try {
        const text = event.text;

        const params = {
            Text: text,
            OutputFormat: "mp3",
            VoiceId: "Joanna",
        };

        // Synthesize speech using Polly
        const command = new SynthesizeSpeechCommand(params);
        const data = await polly.send(command);

        // Generate a unique key for the audio file
        const key = `audio-${Date.now()}.mp3`;

        // Use Upload to stream the audio file to S3
        const upload = new Upload({
            client: s3,
            params: {
                Bucket: "<YOUR-BUCKET-NAME>", //Replace with your bucket name
                Key: key,
                Body: data.AudioStream,
```

```
        // Use Upload to stream the audio file to S3
        const upload = new Upload({
            client: s3,
            params: {
                Bucket: "<YOUR-BUCKET-NAME>", //Replace with your bucket name
                Key: key,
                Body: data.AudioStream,
                ContentType: "audio/mpeg",
            },
        });

        await upload.done(); // Wait for upload to complete

        return {
            statusCode: 200,
            body: JSON.stringify({ message: `Audio file stored as ${key}` }),
        };
    } catch (error) {
        console.error("Error:", error);
        return {
            statusCode: 500,
            body: JSON.stringify({ message: "Internal server error" }),
        };
    }
};
```

Deploy the code changes by clicking on Deploy.

Checking the Output

We have completed with the code configuration of the Lambda function, let's test the function out by creating a test event.

1. Click on **Test** and configure a test event for your Lambda function.

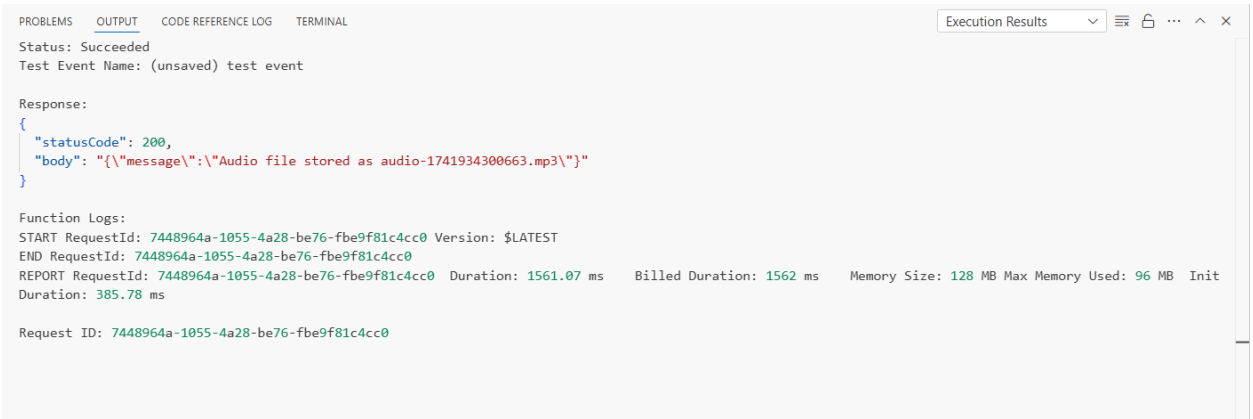
2. Provide a name for your test configuration and in the Event JSON, provide the text you want to be converted to audio in the form:

```JSON
{
        "text":"The text to be converted to Audio"
}
```

3. Leave the rest of configurations as default and click **'Save'**.

4. Click on **Test** button again to invoke the test event.

5. Check the **output.**



6. You can access the audio file by checking it in the previously created S3 bucket and downloading it.