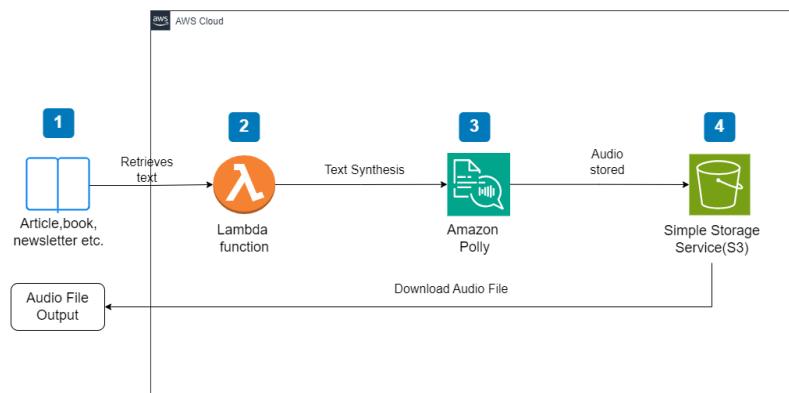


Develop a text narrator using Amazon Polly

Here is the architectural plan of my application:

TEXT NARRATOR USING AWS POLLY



STEP 1

Creating an IAM role

I created an IAM role with suitable policies attached to it in order to access to POLLY service and store the audio output in a S3 bucket using a Lambda function.

SIMPLY | IAM > Roles > Create role

Step 1
① Select trusted entity
Step 2
○ Add permissions
Step 3
○ Name, review, and create

Select trusted entity Info

Trusted entity type

AWS service Allow AWS services like EC2, Lambda, or others to perform actions in this account.

AWS account Allow entities in other AWS accounts belonging to you or a 3rd party to perform actions in this account.

Web identity Allows users federated by the specified external web identity provider to assume this role to perform actions in this account.

SAML 2.0 federation Allow users federated with SAML 2.0 from a corporate directory to perform actions in this account.

Custom trust policy Create a custom trust policy to enable others to perform actions in this account.

Use case
Allow an AWS service like EC2, Lambda, or others to perform actions in this account.

Service or use case

Choose a use case for the specified service.
Use case
 Lambda Allows Lambda functions to call AWS services on your behalf.

Step 2: Add permissions

Edit

Permissions policy summary

Policy name	Type	Attached as
AmazonPollyFullAccess	AWS managed	Permissions policy
AmazonS3FullAccess	AWS managed	Permissions policy
AWSLambdaBasicExecutionRole	AWS managed	Permissions policy

Step 3: Add tags

Add tags - optional

Tags are key-value pairs that you can add to AWS resources to help identify, organize, or search for resources.

No tags associated with the resource.

Add new tag

You can add up to 50 more tags.

IAM > Roles > Create role

- Select trusted entity
- Step 2
- Add permissions
- Step 3
- Name, review, and create**

Name, review, and create

Role details

Role name
Enter a meaningful name to identify this role.

Maximum 64 characters. Use alphanumeric and '+=_@-' characters.

Description
Add a short explanation for this role.

Maximum 1000 characters. Use letters (A-Z and a-z), numbers (0-9), tabs, new lines, or any of the following characters: _+=_, @~\[\]\\$\%^*{}`

Step 1: Select trusted entities

Edit

Trust policy

```
1- [{  "Version": "2012-10-17",  "Statement": [    {      "Effect": "Allow",      "Action": [        "sts:AssumeRole"      ],      "Principal": {        "Service": [          "lambda.amazonaws.com"        ]      }    }  ]}
```

The role has been created

STEP 2

Creating a S3 bucket

Successfully created bucket "mopolybucket-1986"
To upload files and folders, or to configure additional bucket settings, choose View details.

View details X

► Account snapshot - updated every 24 hours All AWS Regions

Storage lens provides visibility into storage usage and activity trends. Metrics don't include directory buckets. Learn more ↗

View Storage Lens dashboard

General purpose buckets Directory buckets

General purpose buckets (1) Info All AWS Regions

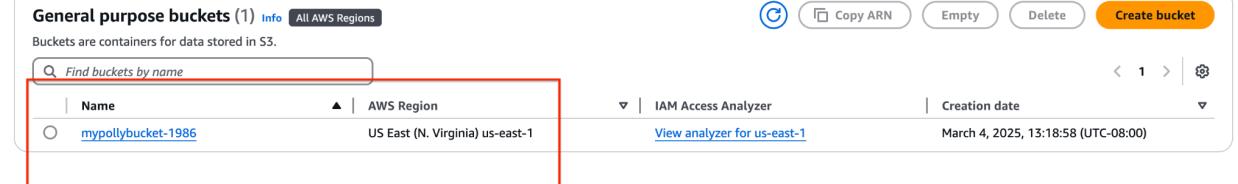
Buckets are containers for data stored in S3.

Name	AWS Region
mopolybucket-1986	US East (N. Virginia) us-east-1

Copy ARN Empty Delete Create bucket

< 1 > ⚙️

IAM Access Analyzer View analyzer for us-east-1 Creation date March 4, 2025, 13:18:58 (UTC-08:00)



The Bucket has been created

STEP 3

Creating a Lambda function

☰ Lambda > Functions > Create function

Create function Info

Choose one of the following options to create your function.

Author from scratch
Start with a simple Hello World example.

Use a blueprint
Build a Lambda application from sample code and configuration presets for common use cases.

Container image
Select a container image to deploy for your function.

Basic information

Function name
Enter a name that describes the purpose of your function.

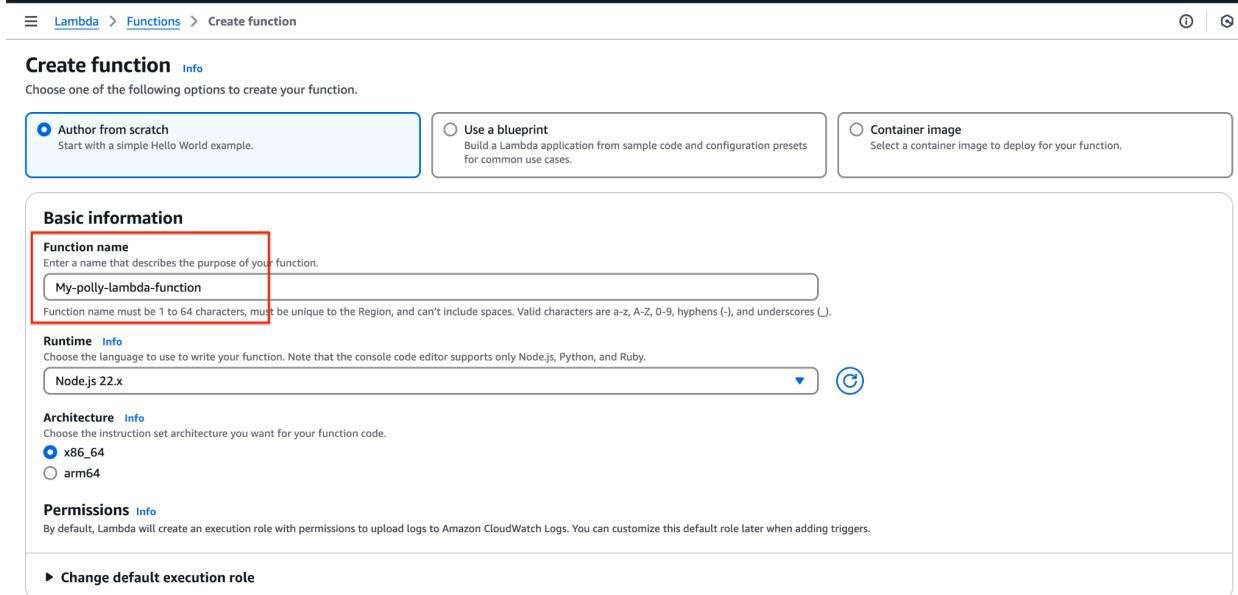
Function name must be 1 to 64 characters, must be unique to the Region, and can't include spaces. Valid characters are a-z, A-Z, 0-9, hyphens (-), and underscores (_).

Runtime Info
Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.
 ⚙️

Architecture Info
Choose the instruction set architecture you want for your function code.
 x86_64
 arm64

Permissions Info
By default, Lambda will create an execution role with permissions to upload logs to Amazon CloudWatch Logs. You can customize this default role later when adding triggers.

▶ Change default execution role



Successfully created the function My-polly-lambda-function. You can now change its code and configuration. To invoke your function with a test event, choose "Test".

My-polly-lambda-function

Function overview [Info](#)

[Diagram](#) [Template](#)

My-polly-lambda-function

Layers (0)

+ Add trigger [+ Add destination](#)

Description

Last modified 12 seconds ago

Function ARN arn:aws:lambda:us-east-1:082706928695:function:My-polly-lambda-function

Function URL [Info](#)

Code Test Monitor Configuration Aliases Versions

My lambda function has been created, while i was creating the lambda function i added the role that we created so lambda can access to S3 and invoke Polly.

STEP 4

Writing the Code

Before i wrote the code i had to modify the index file name from **index.mjs**' file to '**index.js**'

In order for my lambda function to communicate with other service like (POLLY and S3) i had to use AWS SDK.

So i had to include it in my lambda code:

```
const AWS = require('aws-sdk');
const polly = new AWS.Polly();
const s3 = new AWS.S3();
```

I wrote a function that AWS runs for me whenever something happens. It's like a little program that waits for a signal to start working:

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

The Objective was, when a message is received by the lambda function it has to turn it into speech. I was able to choose how the speech will sound and in which format:

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

I sent the text to Polly and asked it to turn it into speech. Polly did its magic and gave me back the speech as data.

I then saved this speech in my S3 storage:

```
const data = await polly.synthesizeSpeech(params).promise();
const key = `audio-${Date.now()}.mp3`;
const s3Params = {
  Bucket: 'your-bucket-name', // Replace with your S3 bucket name
  Key: key,
  Body: data.AudioStream,
  ContentType: 'audio/mpeg'
};
await s3.upload(s3Params).promise();
```

I made a message saying the speech has been saved successfully with its special name in my storage. If something goes wrong, there is an error message.

```
const outputMessage = `The audio file has been successfully stored in the S3 bucket by the name ${key}`;
return {
  statusCode: 200,
  body: JSON.stringify({ message: outputMessage })
};
} catch (error) {
  console.error('Error:', error);
  return {
    statusCode: 500,
    body: JSON.stringify({ message: 'Internal server error' })
  };
}
```

STEP 5

Deployin the Code

Here is the complete Lambda function code:

```
const AWS = require('aws-sdk');
const polly = new AWS.Polly();
const s3 = new AWS.S3();
exports.handler = async (event) => {
  try {
    // Extract text input from the event
    const text = event.text;

    // Specify parameters for Polly
    const params = {
      Text: text,
      OutputFormat: 'mp3',
      Voiceld: 'Joanna' // You can change this to the desired voice
    };
    // Synthesize speech using Polly
    const data = await polly.synthesizeSpeech(params).promise();
    // Generate a unique key for the audio file
    const key = `audio-${Date.now()}.mp3`;
    // Specify parameters for S3
    const s3Params = {
      Bucket: 'your-bucket-name', // Replace with your S3 bucket name
      Key: key,
      Body: data.AudioStream,
      ContentType: 'audio/mpeg'
    };
    // Upload audio file to S3
    await s3.upload(s3Params).promise();
    const outputMessage = `The audio file has been successfully stored in the S3 bucket by the name ${key}`;
    return {
      statusCode: 200,
      body: JSON.stringify({ message: outputMessage })
    };
  } catch (error) {
    console.error('Error:', error);
    return {
      statusCode: 500,
      body: JSON.stringify({ message: 'Internal server error' })
    };
  }
}
```

};

I deployed the code changes by clicking on Deploy

The screenshot shows the AWS Lambda Code source interface. On the left, the EXPLORER panel shows a project named 'MY-POLLY-LAMBDA-FUNCTION' containing an 'index.js' file. In the center, the code editor displays the 'index.js' file with the following content:

```
JS index.js > ...
6   exports.handler = async (event) => {
7     const s3Params = {
8       Key: key,
9       Body: data.AudioStream,
10      ContentType: 'audio/mpeg'
11    };
12
13    // Upload audio file to S3
14    await s3.upload(s3Params).promise();
15
16    const outputMessage = `The audio file has been successfully stored in the S3 bucket by the name ${key}`;
17
18    return {
19      statusCode: 200,
20      body: JSON.stringify({ message: outputMessage })
21    };
22  } catch (error) {
23    console.error('Error:', error);
24    return {
25      statusCode: 500,
26      body: JSON.stringify({ message: 'Internal server error' })
27    };
28  }
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48 };
```

Below the code editor, the 'DEPLOY [UNDEPLOYED CHANGES]' section indicates 'You have undeployed changes.' It contains two buttons: 'Deploy (D)' and 'Test (T)'. The 'Deploy' button is highlighted with a red box.

STEP 6

Checking the output

I tested and configured an event for my lambda function

The screenshot shows the AWS Lambda Functions interface. On the left, the EXPLORER panel shows a project named 'MY-POLLY-LAMBDA-FUNCTION' containing an 'index.js' file. In the center, the code editor displays the 'index.js' file with the same content as before. Below the code editor, the 'TEST EVENTS [SELECTED: ...]' section shows a test event named 'mylambdafunctiontest' selected. It contains two buttons: 'Deploy (D)' and 'Test (T)'. The 'Test' button is highlighted with a red box.

On the right, a modal window titled 'Create new test event' is open. It includes fields for 'Event Name' (set to 'mylambdafunctiontest'), 'Event sharing settings' (with 'Private' selected), and a 'Template - optional' dropdown set to 'mylambdafunctiontest'. A large text area labeled 'Event JSON' contains the following JSON:

```
1  {
2    "text": "The text to be converted to Audio"
3  }
```

A message at the bottom of the modal says 'Test event is saved successfully.'

I provided a name for my test configuration and in the Event JSON, I provided the text to be converted to audio.

"text": "The text to be converted to Audio"