**Building a Serverless Text-to-Speech Application with Amazon Polly**

[Polly]

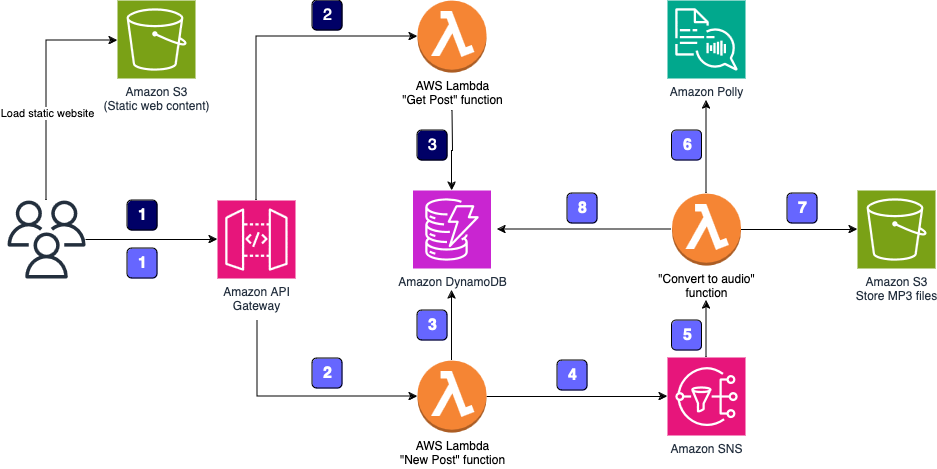
**Introduction**

Serverless application, means that you do not need to work with servers — no provisioning, no patching, no scaling. AWS automatically takes care of this, allowing you to focus on your application.

**Project Description**

This application provides two methods – one for **sending information** about a new post, that will be converted into an MP3 file, and one for **retrieving information** about the post (including a link to the MP3 file stored in an Amazon S3 bucket). I will expose both as RESTful web services through Amazon API Gateway.

**Architecture – Overview**



**How it works**

**When the application sends information about new posts:**

The information is received by the RESTful web service exposed by Amazon API Gateway. This web service is invoked by a static webpage hosted on Amazon Simple Storage Service (Amazon S3).

Amazon API Gateway triggers an AWS Lambda function, New Post, that is responsible for initializing the process of generating MP3 files.

The Lambda function inserts information about the post into an Amazon DynamoDB table, where information about all posts is stored.

To run the whole process asynchronously, I will use Amazon Simple Notification Service (Amazon SNS) to decouple the process of receiving information about new posts and starting their audio conversion.

Another Lambda function, Convert to Audio, is subscribed to the SNS topic and is triggered whenever a new message appears (which means that a new post should be converted into an audio file).

The Convert to Audio Lambda function uses Amazon Polly to convert the text into an audio file in the specified language (the same as the language of the text).

The new MP3 file is saved in a dedicated S3 bucket.

Information about the post is updated in the DynamoDB table. The URL to the audio file stored in the S3 bucket is saved with the previously stored data.

**When the application retrieves information about posts:**

The RESTful web service is deployed using Amazon API Gateway. Amazon API Gateway exposes the method for retrieving information about posts. These methods contain the text of the post and the link to the S3 bucket where the MP3 file is stored. The web service is invoked by a static webpage hosted on Amazon S3.

Amazon API Gateway invokes the Get Post Lambda function, which deploys the logic for retrieving the post data.

The Get Post Lambda function retrieves information about the post (including the reference to Amazon S3) from the DynamoDB table and returns the information.

**Launching the CloudFormation stack**

The CloudFormation template provisions the following resources:

* IAM Role.
* Amazon DynamoDB table.
* AWS Step Functions State Machine.

**Create an Amazon DynamoDB table to store data**

The application stores information about the posts, including the text and URL of the MP3 file, in Amazon DynamoDB.

**A screenshot of a computer

Description automatically generated**

**Create an Amazon S3 bucket**

To store all audio files created by the application.

A screenshot of a computer

Description automatically generated

**Create an SNS topic - Integrating the two processes**

Amazon SNS sends the message about the new post from the first function to the second function these Lambda functions use the same SNS TOPIC ARN.

**A screen shot of a computer

Description automatically generated**

The logic of converting a post (text) into an audio file is split into two AWS Lambda functions.

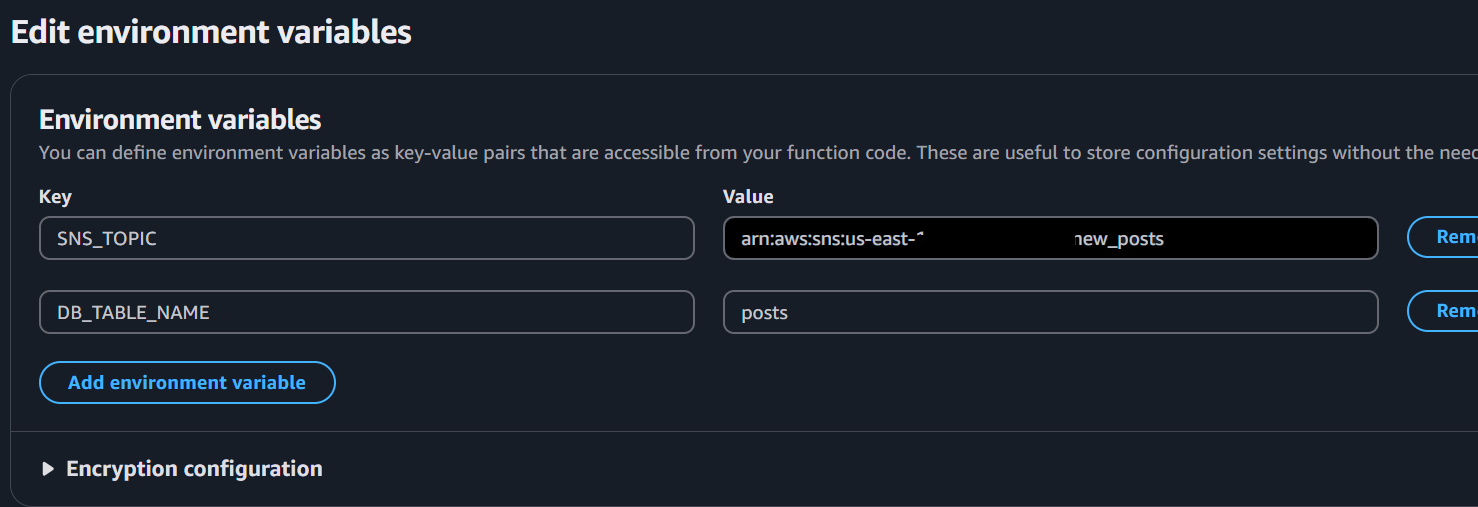
First, it allows the application to use asynchronous calls so that the user who sends a new post to the application immediately receives the ID of the new DynamoDB item, so it knows what to ask for later without having to wait for the conversion to finish.

Second, the system uses a Lambda function to convert the posts.

**Create a new post Lambda function [1st Function]**

The first Lambda function is the entry point for the application. It receives information about new posts that should be converted into audio files.

The Lambda function needs to know the name of the DynamoDB table and the SNS topic. To provide these values, you use environment variables. This is an excellent way to pass information to a function without hard-coding values into the function itself.



**Test Function works**

**A screenshot of a computer

Description automatically generated**

**The Lambda function does the following:**

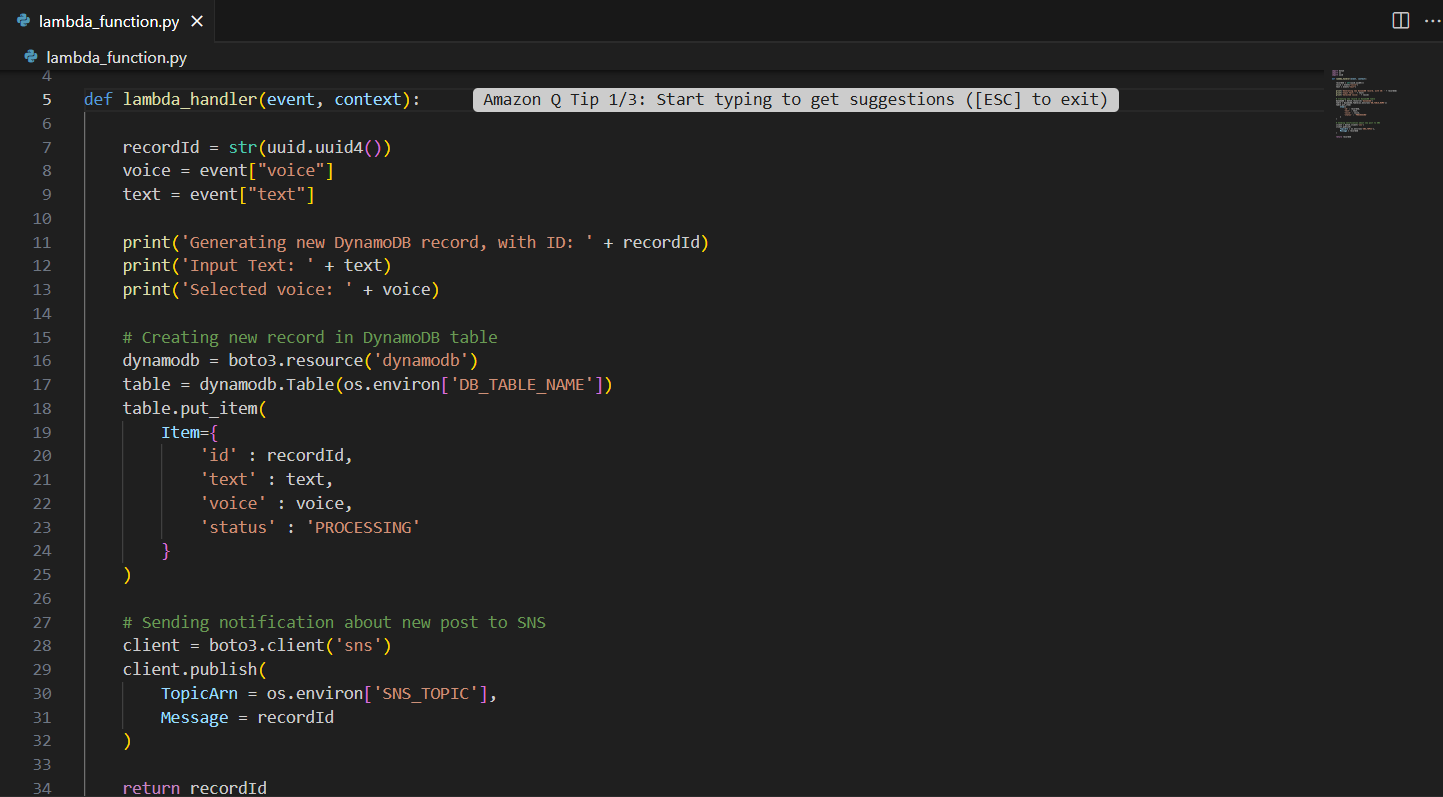
Voice: One of dozens of voices that are supported by Amazon Polly.

Text: The text of the post that we want to convert into an audio file.

Creates a new record in the DynamoDB table with information about the new post

Publishes information about the new post to SNS (the ID of the DynamoDB item/post ID is published there as a message)

Returns the ID of the DynamoDB item to the user.



**Create a convert to audio Lambda function [ 2nd Function]**

This Lambda function that converts text that is stored in the DynamoDB table into an audio file.

**The Lambda function does the following:**

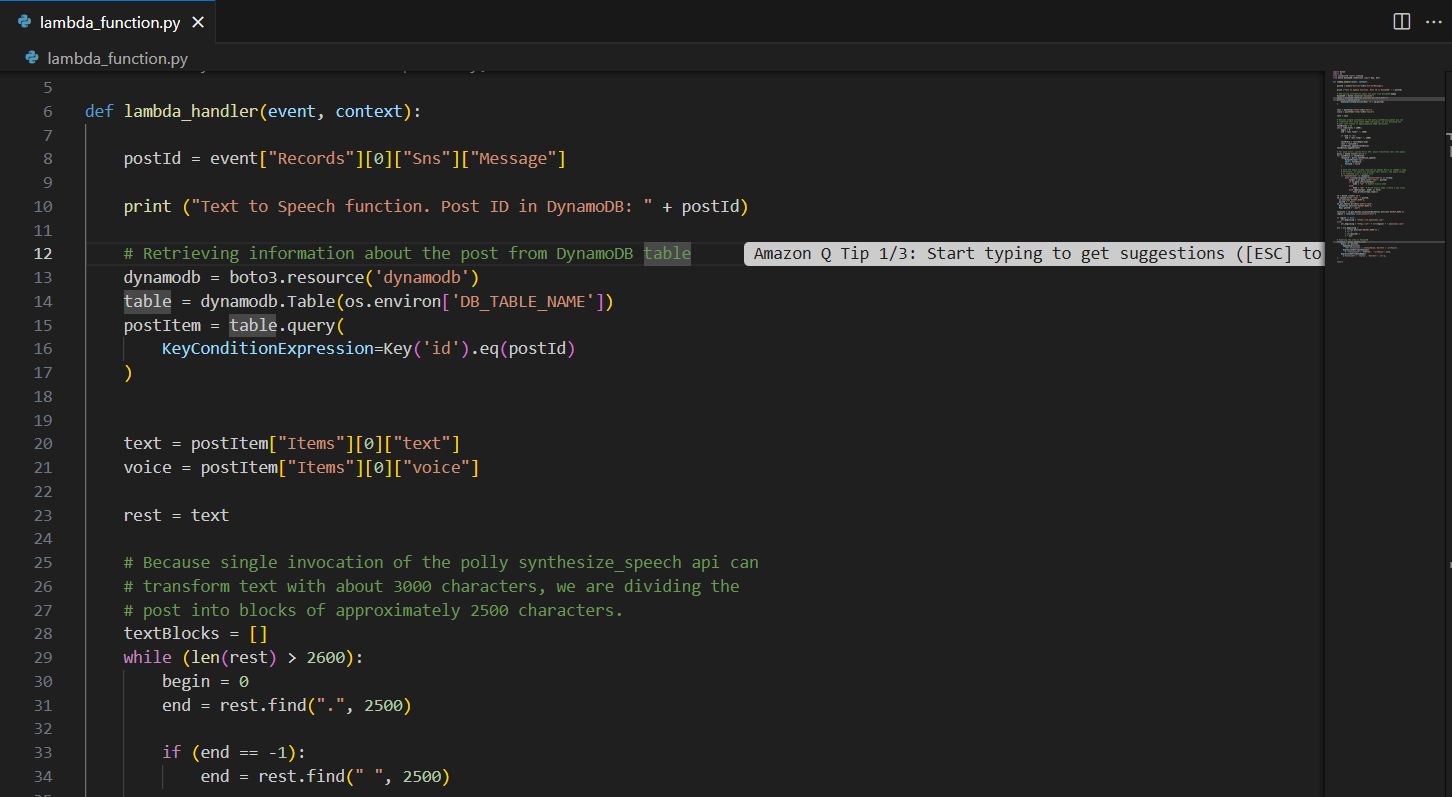
Retrieves the ID of the DynamoDB item (post ID) which should be converted into an audio file from the input message (SNS event)

Retrieves the item from DynamoDB.

Converts the text into an audio stream.

Places the audio (MP3) file into an S3 bucket.

Updates the DynamoDB table with a reference to the S3 bucket and the new status.

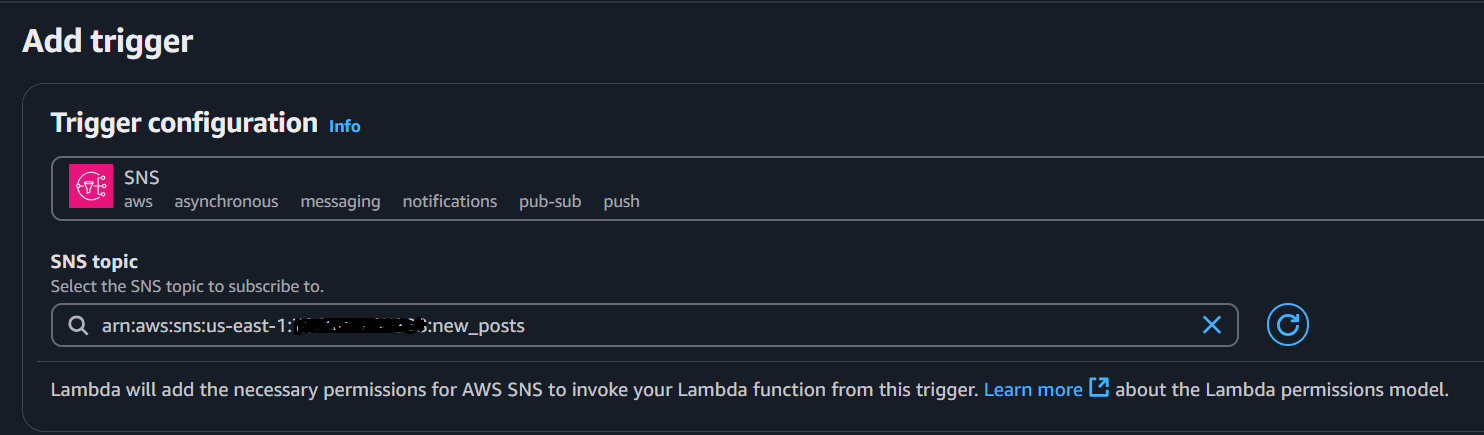


This Lambda function needs to be told which services it can interact with via **Environment variables.**

A screenshot of a computer

Description automatically generated

A function trigger will be added to trigger automatically when a message is sent to the SNS topic.



**Function testing**

I will Manually trigger the *New Post* Lambda function and see if the functions can communicate.

Stores data in DynamoDB and send a message to the SNS topic

SNS triggers the Convert To Audio function, which uses Polly to create an audio file and store it in the S3 bucket

Manual triggers - Lambda

A screenshot of a computer

Description automatically generated

Entries in DynamoDB

A screenshot of a computer

Description automatically generated

SNS triggers ConvertToAudio function

A screenshot of a computer

Description automatically generated

Polly creates audio file and stores it my S3 bucket

A screenshot of a computer

Description automatically generated

CloudWatch>>log groups>>Log event indicate that the function has been invoked.

A screenshot of a computer

Description automatically generated

**Creating a get post lambda function [3rd function]**

This Lambda function provides a method for retrieving information about posts from the database.

A screen shot of a computer

Description automatically generated

This function gets the post ID (the DynamoDB item ID) and it retrieves all the information (including the S3 link to the audio file if it exists) and then returns it.

It will also need the name of the DynamoDB table as an Environment variable for the function.

**Expose the Lambda function as a RESTful web service**

I will expose the application logic as a RESTful web service so it can be invoked easily using a standard HTTP protocol using Amazon API Gateway.

**Creating a Rest API**

I will create two http methods one for the **POST** method to invoke the PostReader\_NewPost Lambda function.

A screenshot of a computer

Description automatically generated

And for the **GET** method the API invokes the Post\_GetPost Lambda function

A screen shot of a computer

Description automatically generated

I will also enable CORS; this allows web browsers to make requests to my API/server from different origins.

The PostReader\_GetPost Lambda function expects to receive input data in JSON format, so the API needs to be configured to map the parameter into this format. To do this, you can add mapping to the Integration Request configuration.

A black and white background with lines

Description automatically generated with medium confidence

**Create a Serverless User Interface**

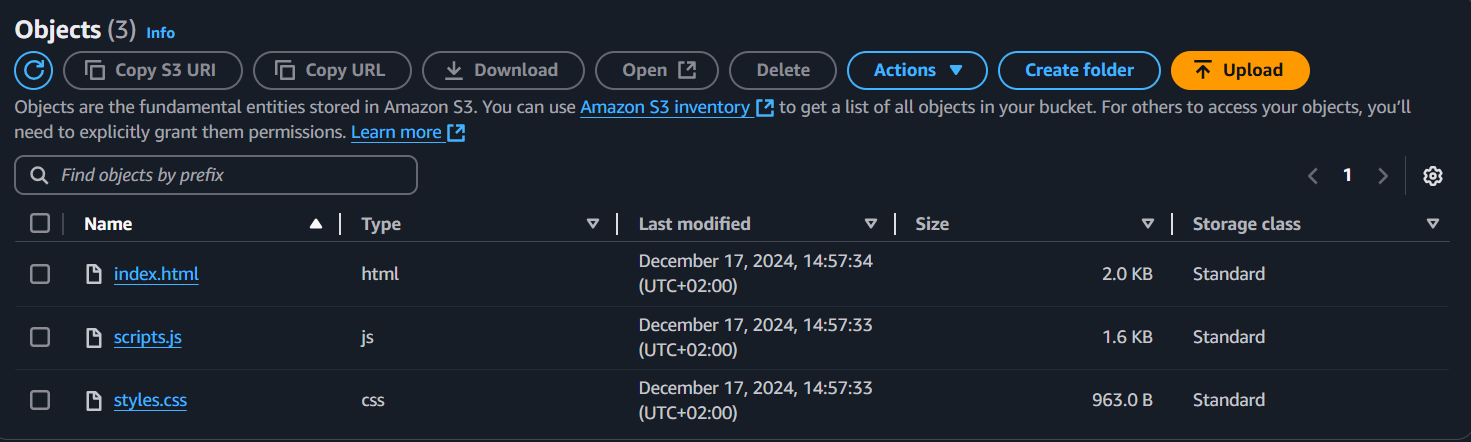
I will deploy a small web page on Amazon S3 for hosting static web pages. This web page uses JavaScript to connect to the API and it provides text-to-speech functionalities in a web page.

Changedthe API Endpoint before uploading, this is used to interact with the application.

A computer screen shot of a program code

Description automatically generated

Items uploaded.



S3 Endpoint URL used to connect to the website

A screenshot of a computer

Description automatically generated

**Conclusion | Re-Cap**

Resources created for this project were

Created an Amazon DynamoDB table to store data

Created an Amazon API Gateway RESTful API

Created AWS Lambda functions triggered by API Gateway

Connected AWS Lambda functions with Amazon Simple Notification Service (SNS)

Used Amazon Polly to synthesize speech in a variety of languages and voices

That enabled the application to convert text into speech.

The application is completely serverless.

There are no servers to maintain or patch.

By default, the application is highly available because AWS Lambda, Amazon API Gateway, Amazon S3, and Amazon DynamoDB use multiple Available Zones

**Done!**