A Training Report

On

“AWS project:-polly”

Submitted to the Rajasthan Technical University, Kota

In partial fulfillment of the requirement for the degree of

MASTER OF COMPUTER APPLICATIONS

Submitted by:-

Nimisha vilayatarani

Roll no.: 18CPGXX264

Name of Internal Guide	**Name of Training in-charge**
Mrs. Mamta Sharma	**Kawaljeet singh**

S. S. Jain Subodh P.G. College

Affiliated to

Rajasthan Technical University, Kota

MCA – Batch (2018-2020)

April, 2020

Ethan's™
Learn from experts

Certificate of Completion

This is to certify that

NIMISHA VIJAYATRANI

has successfully Completed AMAZON WEB SERVICES

from JULY-2019 to SEP-2019.

25th September, 19
Date of Issue

IKVOA DAC
ISO 9001-2015 BY DAC

ETHANS TECH SOLUTIONS LLP PUNE
Signature

Ethans Tech Solutions LLP

Ethan
Learn from expe

Date: 29/Sep/19

To whom it may concern

This is to certify that Ms. Nimisha Vilayatrani has joined Amazon Web Services training program - Ethans Tech, Baner Branch under the supervision of Kawaljeet Singh (HOD - Amazon Web Services) on Weekends (Saturday & Sunday) which started from 23th June 2019 and ends on 29th September 2019. The Total Duration will be 70Hrs (includes the Certification & Project Preparation)

ETHANS TECH SOLUTIONS LLP PUNE

Sincerely,

Mayank Miglani
Ethans Tech

Regd Address: A2-305, Royal Castle
Opp Royal Court Thergaon, Pune 411033
Cell: 9527354004/8550992001
E-Mail ID: training@ethans.co.in
Website: www.ethans.co.in

Acknowledgment

I express my deep sense of gratitude to my respected and learned guide,

Mrs. Mamta Sharma for her valuable help and guidance. I am thankful for the encouragement she has given me in the completion of my project.

I am also grateful to our respected **Prof. K.B. Sharma**, Director and **Dr. Leena Bhatia Mam,** Head of Department for allowing me to utilize all the necessary resources of the institution to complete my project. I am also thankful to all the other faculty and staff members of our department for their kind co-operation and help.

Lastly, I would like to express my deep appreciation towards my classmates and indebtedness to my parents for providing me the moral support and encouragement.

Date :

Nimisha vilayatarani

Class - MCA - VI

Roll No. 18CPGXX264

Table of Content

• Introduction to Project

Amazon Polly provides speech synthesis functionality that overcomes those challenges, allowing you to focus on building applications that use text-to-speech instead of addressing interpretation challenges.

Amazon Polly turns text into lifelike speech. It lets you create applications that talk naturally, enabling you to build entirely new categories of speech-enabled products. Amazon Polly is an Amazon AI service that uses advanced deep learning technologies to synthesize speech that sounds like a human voice. It currently includes 47 lifelike voices in 24 languages, so you can select the ideal voice and build speech-enabled applications that work in many different countries.

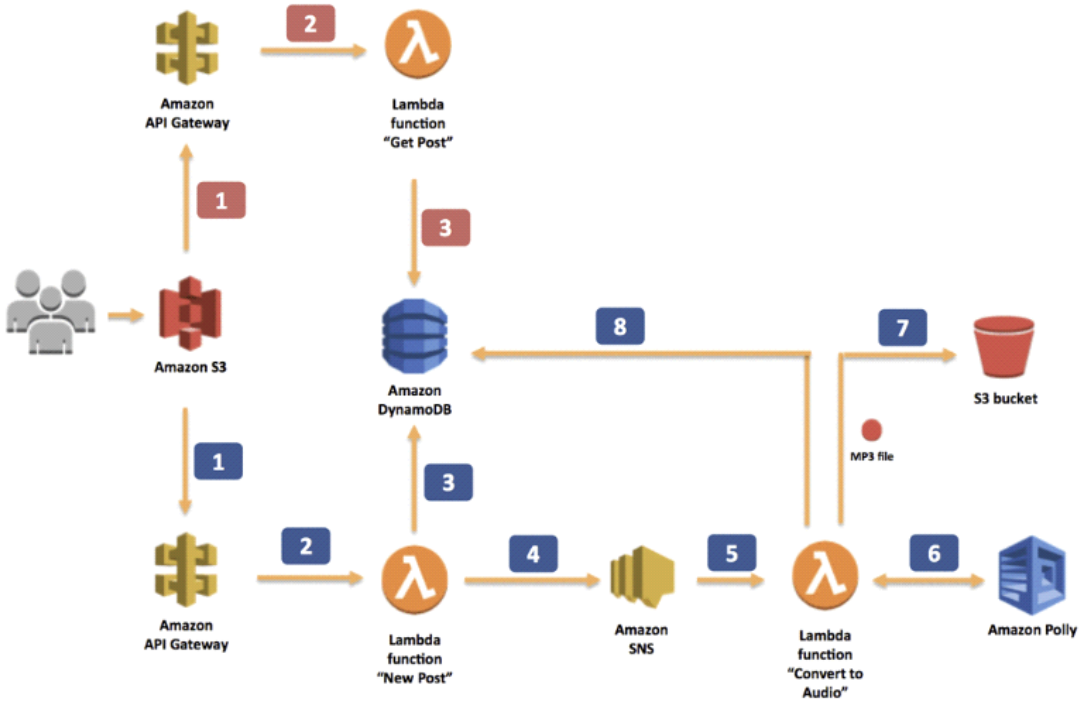
In addition, Amazon Polly delivers the consistently fast response times required to support real-time, interactive dialog. You can cache and save

Polly's audio files for offline replay or redistribution. (In other words, what you convert and save is yours. There are no additional text-to-speech charges for using the speech.) And Polly is easy to use. You simply send the text you want to convert into speech to the Amazon Polly API. Amazon Polly immediately returns the audio stream to your application so that your application can play it directly or store it in a standard audio file format such as an MP3.

In this, we create a basic, serverless application that uses Amazon Polly to convert text to speech. The application has a simple user interface that accepts text in many different languages and then converts it to audio files which you can play from a web browser. We'll use blog posts, but you can use any type of text. For example, you can use the application to read recipes while you are preparing a meal, or news articles or books while you're driving or riding a bike.

The application's architecture

- The following diagram shows the application architecture. It uses a serverless approach, which means that we don't need to work with servers – no provisioning, no patching, no scaling. The Cloud automatically takes care of this, allowing us to focus on our application.



When the application sends information about new posts:

- The information is received by the restful web service exposed by Amazon API Gateway. In our scenario, this web service is invoked by a static webpage hosted on Amazon Simple Storage Service (Amazon S3).
- Amazon API Gateway sets off a dedicated Lambda function, “New Post,” which is responsible for initializing the process of generating MP3 files.
- The Lambda function inserts information about the post into a DynamoDB table, where information about all posts is stored.
- To run the whole process asynchronously, we use Amazon SNS to decouple the process of receiving information about new posts and starting their conversion.
- Another Lambda function, “Convert to Speech,” is subscribed to our SNS topic whenever a new message appears (which means that a new post should be converted into an audio file). This is the trigger.
- The “Convert to Speech” 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. Then, the reference (URL) to the S3 bucket is saved with the previously

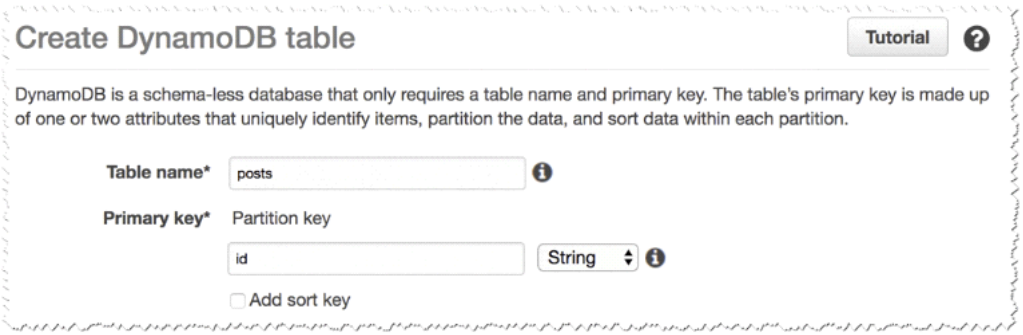
stored data.

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. In our scenario, this 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.

Creating a DynamoDB table

We store information about posts, including the text and URL for the MP3 file, on DynamoDB. From the DynamoDB console we create a single table, which we call “posts.” Our primary key (id) is a string, which the “New Post” Lambda function creates when new records (posts) are inserted into a database.



id	status	text	voice	url
5efdb728-a193-	UPDATED	Hallo, mein N...	Marlene	https://s3-eu-...
763a841c-9507	UPDATED	Hello! My nam...	Joanna	https://s3-eu-...
b98fdc51-563b-	UPDATED	Cześć! Jeste...	Maja	https://s3-eu-...
7366bec3-76ff-4	UPDATED	Hola! Mi nom...	Enrique	https://s3-eu-...

id – The ID of the post

status – UPDATED or PROCESSING, depending on whether an MP3 file has already been created

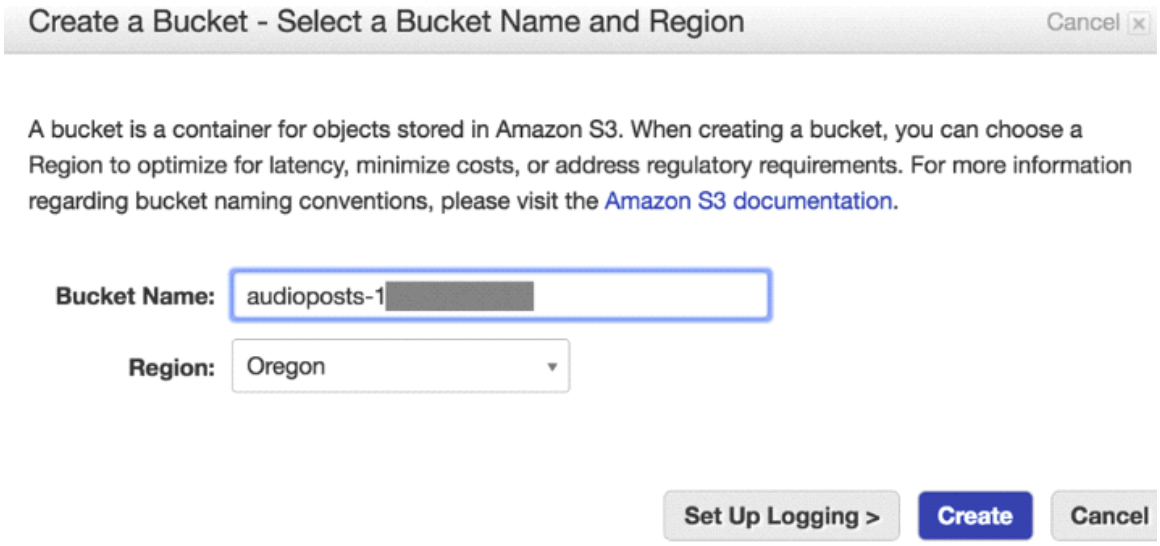
text – The post's text, for which an audio file is being created

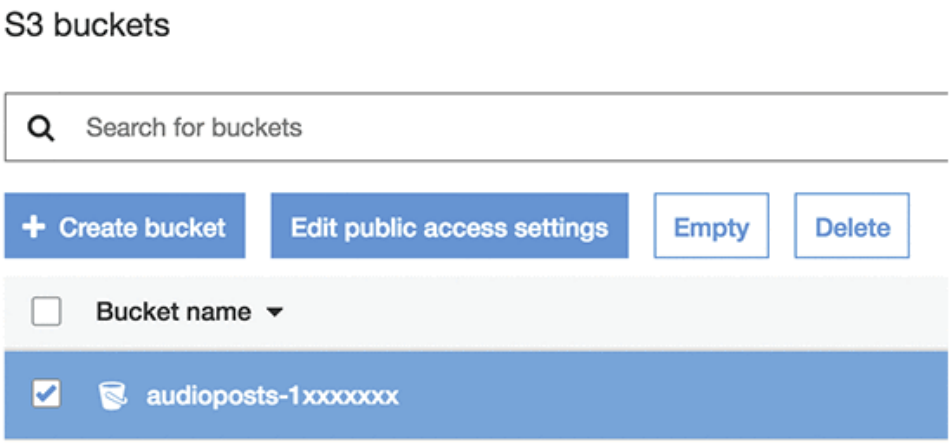
url – A link to an S3 bucket where an audio file is being stored

voice – The Amazon Polly voice that was used to create audio file.

Creating an S3 bucket

We also need to create an S3 bucket to store all audio files created by the application. To do this, you need to go to the S3 console where you will find an option to create a new bucket. You can choose any name for the bucket as long as it's globally unique.





Because we will want to make our audio files in S3 public, we will also need to configure our new S3 bucket and allow this kind of operation. In S3 console, check your new created bucket and click on “Edit public access settings”. In the new popup window just click ‘Save’ and confirm the action.

Creating an SNS topic

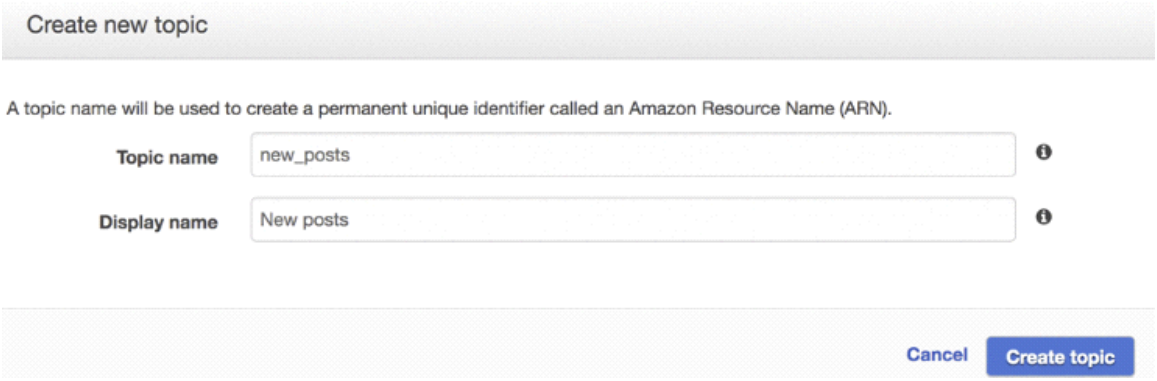
As you probably noticed in our architecture diagram, we have split the logic of converting a post (text) into an audio file into two Lambda functions. We did this for a couple of reasons. First, it allows our application to use asynchronous calls so that the user who sends a new post to the application receives the ID of the new DynamoDB item; so it knows what to ask for later; and to eliminate waiting for the conversion to finish. With small posts, the process of converting to audio files can take milliseconds, but with bigger posts (100,000 words or more), converting the text can take a bit longer. In other use cases, when we want to do real-time streaming, size isn’t a problem, because Amazon Polly starts to stream speech back as soon as the first bytes are available.

The second reason is that we use a Lambda function, which allows a single execution to run as long as 5 minutes. This should be more than enough time to convert our posts. In the future we might want to convert something bigger. In that case, we might want to use AWS Batch instead of Lambda. Decoupling these two parts of the application makes this change much easier.

When we have two components (in our case two Lambda functions) we

need to integrate them. In other words, the second one needs to know when to start. You could do this in many different ways. In our case, we will use Amazon SNS. It sends the message about the new post from the first function to the second one.

So let's create a simple SNS topic. We can do it from the SNS console, where you will find a button for creating a new topic. Let's call it new_posts.



Creating an IAM role

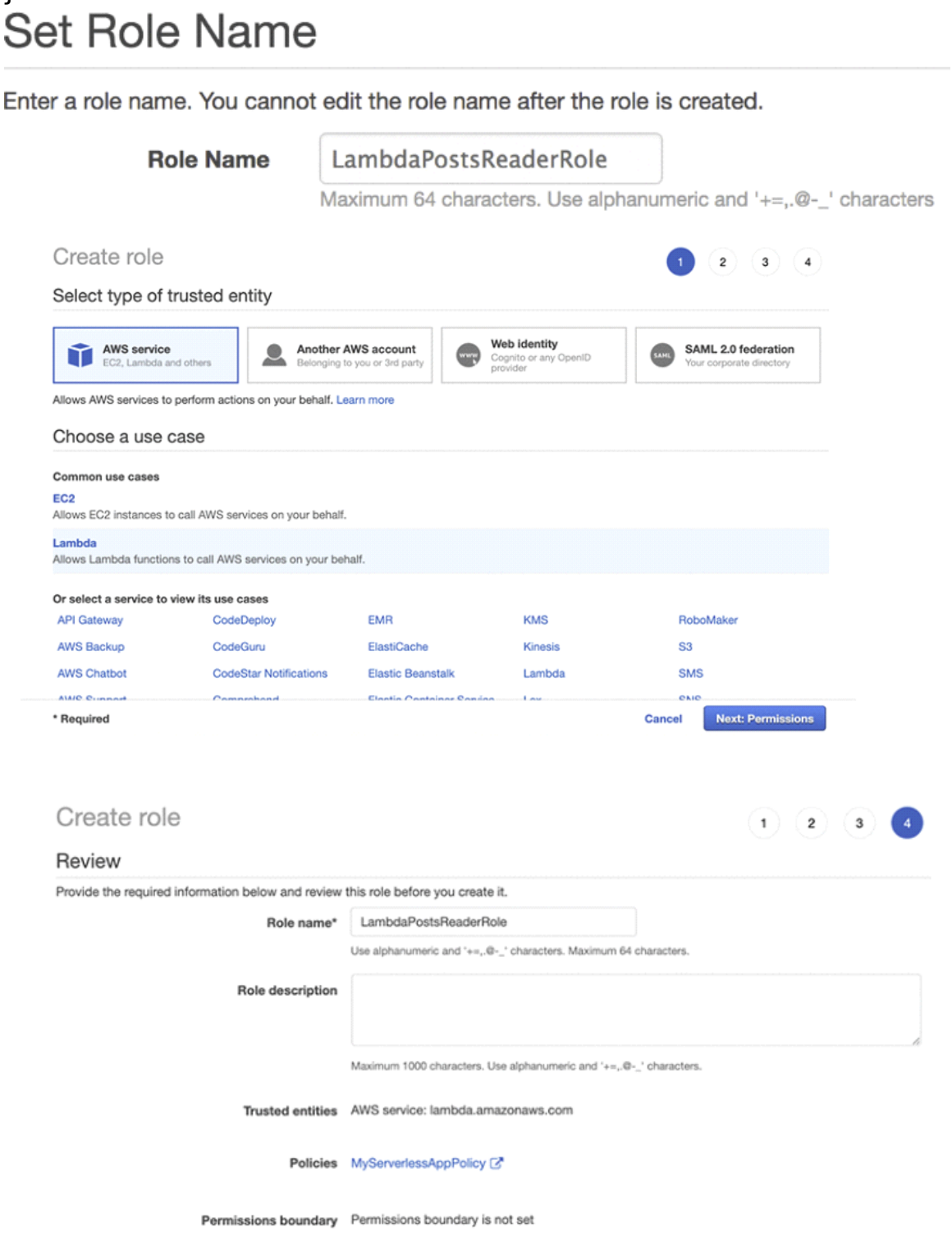
Before we dive into creating Lambda functions, we need to create an IAM role for the functions. The role specifies which AWS services (APIs) the functions can interact with. We will create one role for all three functions.

In the IAM console, find the Policies tab and then press the Create Policy button to open a wizard for creating a new policy. Click on the JSON tab and paste the following script.

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid": "Perm1",
         "Effect": "Allow",
         "Action": [
            "polly:SynthesizeSpeech",
            "s3:GetBucketLocation",
            "logs:CreateLogGroup",
            "logs:CreateLogStream",
            "logs:PutLogEvents"
```

```
            ],
            "Resource": "*"
        },
        {
            "Sid": "Perm2",
            "Effect": "Allow",
            "Action": [
                "dynamodb:Query",
                "dynamodb:Scan",
                "dynamodb:PutItem",
                "dynamodb:UpdateItem"
],
            "Resource":
"arn:aws:dynamodb:REGION:ACCOUNT_ID:table/DYNAMODB_TABLE
_NAME"
        },
        {
            "Sid": "Perm3",
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:PutObjectAcl",
                "s3:GetBucketLocation"
            ],
            "Resource": "arn:aws:s3:::BUCKET_NAME/*"
        },
        {
            "Sid": "Perm4",
            "Effect": "Allow",
            "Action": [
                "sns:Publish"
            ],
            "Resource":
"arn:aws:sns:REGION:ACCOUNT_ID:SNS_TOPIC_NAME"
        }
    ]
```

}

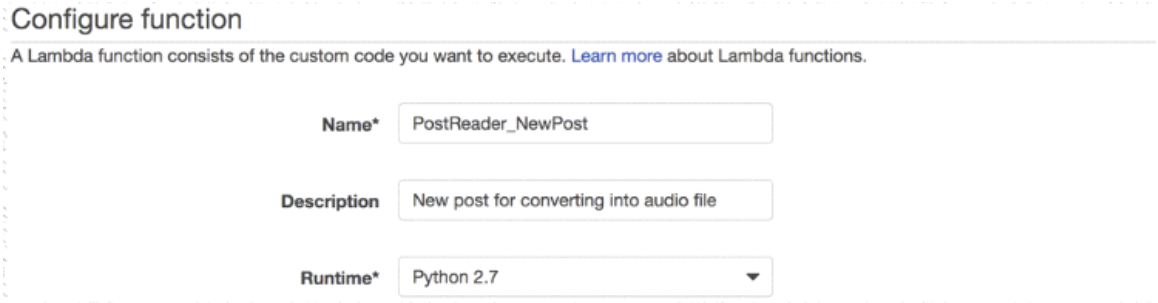


Creating the “New Post” Lambda function

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

In the Lambda console, you will see a button for creating a new Lambda function. Let's call it PostReader_NewPost. For Runtime, we choose Python 2.7. For now, we don't configure any triggers.

s shown in the following code, this Lambda function does the following:



Retrieves two input parameters:

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

```
import boto3
import os
import uuid

def lambda_handler(event, context):

    recordId = str(uuid.uuid4())
    voice = event["voice"]
    text = event["text"]

    print('Generating new DynamoDB record, with ID: ' + recordId)
    print('Input Text: ' + text)
    print('Selected voice: ' + voice)
```

```
#Creating new record in DynamoDB table
dynamodb = boto3.resource('dynamodb')
table = dynamodb.Table(os.environ['DB_TABLE_NAME'])
table.put_item(
    Item={
        'id' : recordId,
        'text' : text,
        'voice' : voice,
        'status' : 'PROCESSING'
    }
)

#Sending notification about new post to SNS
client = boto3.client('sns')
client.publish(
    TopicArn = os.environ['SNS_TOPIC'],
    Message = recordId
)
```

Still in this same wizard, we assign the IAM role that we created for the Lambda functions.


```
    return recordId
```

In addition, the New Post Lambda function needs to know the name of the DynamoDB table and the SNS topic. To provide these values, we use the following environment variables:

SNS_TOPIC – the Amazon Resource Name (ARN) of the SNS topic we created

DB_TABLE_NAME – the name of the DynamoDB table (in our case, it's posts)

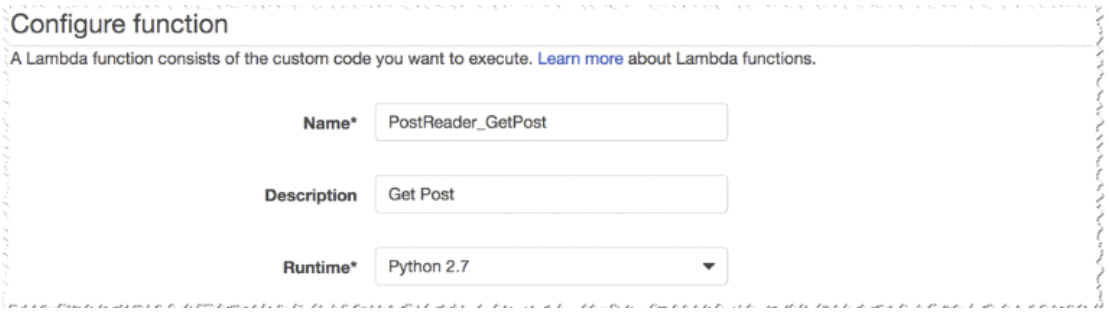
You will find Environment variables section just below your code.

Creating the "Convert to Audio" Lambda function

Now let's create the Lambda function that converts text that is stored in a DynamoDB table into an audio file, "Convert to Audio."

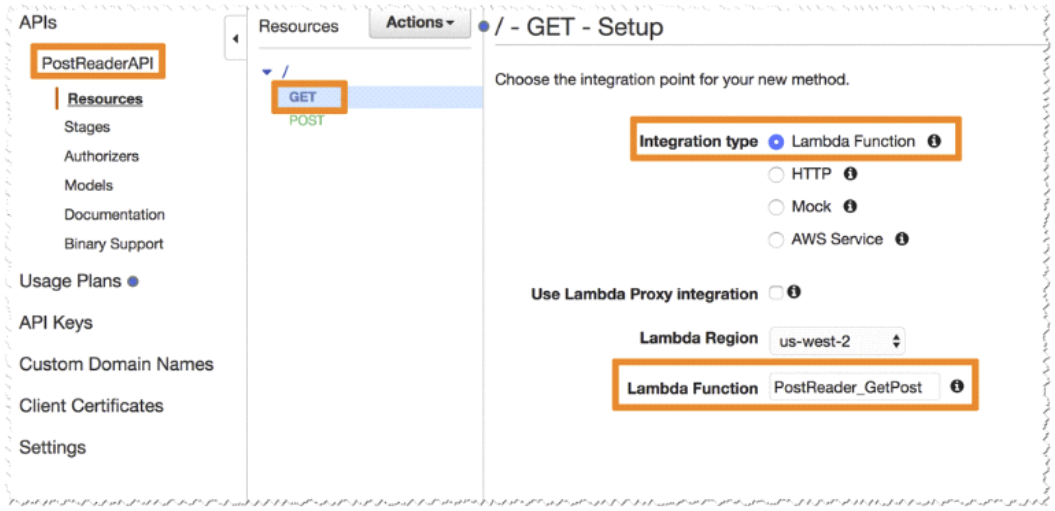
In the first step of the wizard, we specify the SNS topic that we created. This time, we configure and enable a trigger. Whenever our SNS topic receives a new message, it executes this function.

screen.

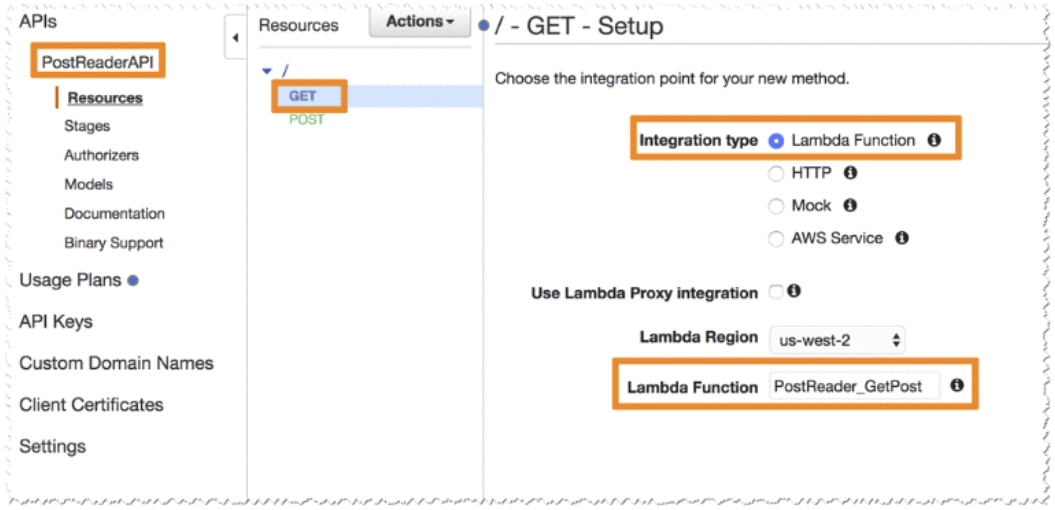


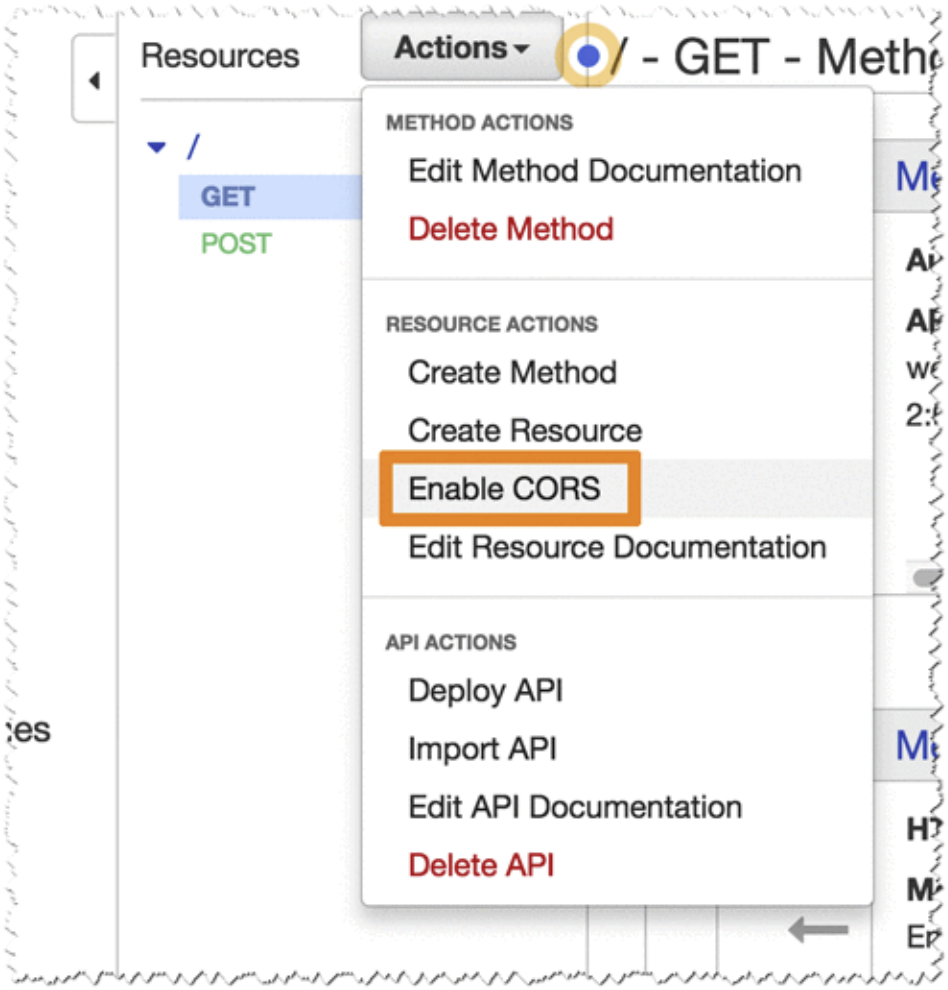

```python
import boto3
import os
from boto3.dynamodb.conditions import Key, Attr

def lambda_handler(event, context):

    postId = event["postId"]

    dynamodb = boto3.resource('dynamodb')
    table = dynamodb.Table(os.environ['DB_TABLE_NAME'])

    if postId=="*":
        items = table.scan()
    else:
        items = table.query(
            KeyConditionExpression=Key('id').eq(postId)
        )

    return items["Items"]
```

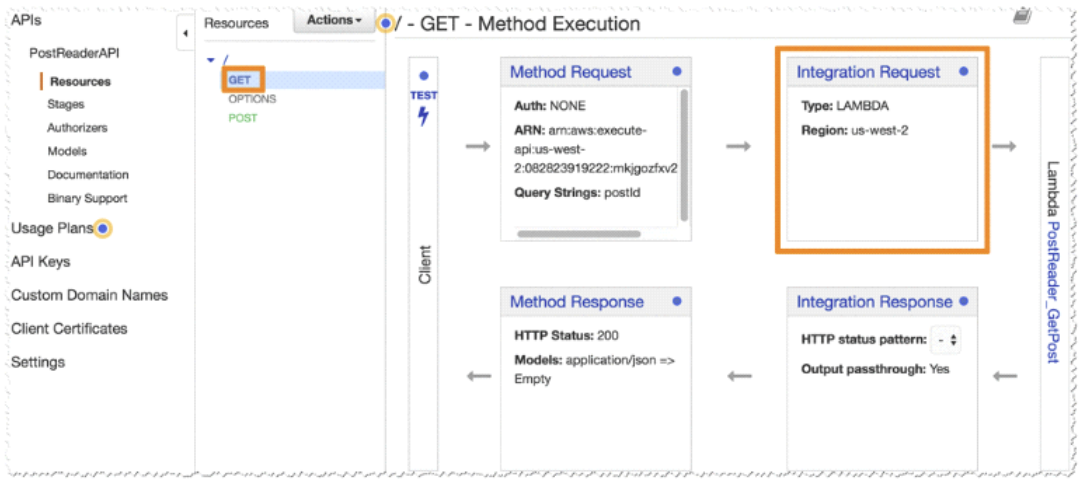

Custom Domain Names
Client Certificates
Settings
Lambda Function
PostReader_NewPost


APIs
PostReaderAPI
Resources
Stages
Authorizers
Models
Documentation
Binary Support
Usage Plans
API Keys
Custom Domain Names
Client Certificates
Settings
Resources
Actions
/
GET
POST
/ - GET - Setup
Choose the integration point for your new method.
Integration type
Lambda Function
HTTP
Mock
AWS Service
Use Lambda Proxy integration
Lambda Region
us-west-2
Lambda Function
PostReader_GetPost


Custom Domain Names
Client Certificates
Settings
Lambda Function
PostReader_NewPost

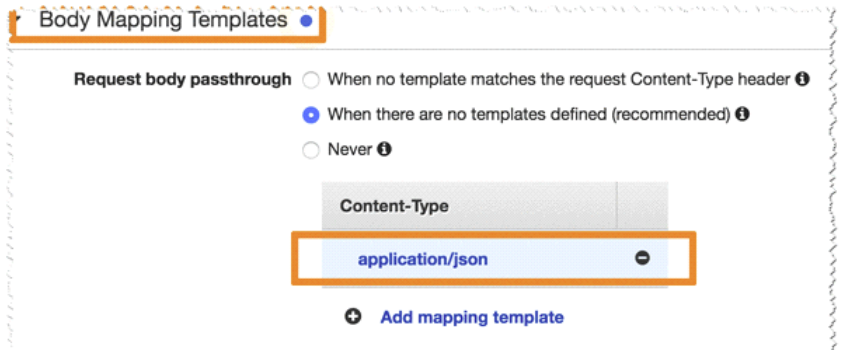

APIs
PostReaderAPI
Resources
Stages
Authorizers
Models
Documentation
Binary Support
Usage Plans
API Keys
Custom Domain Names
Client Certificates
Settings
Resources
Actions
/
GET
POST
/ - GET - Setup
Choose the integration point for your new method.
Integration type
Lambda Function
HTTP
Mock
AWS Service
Use Lambda Proxy integration
Lambda Region
us-west-2
Lambda Function
PostReader_GetPost

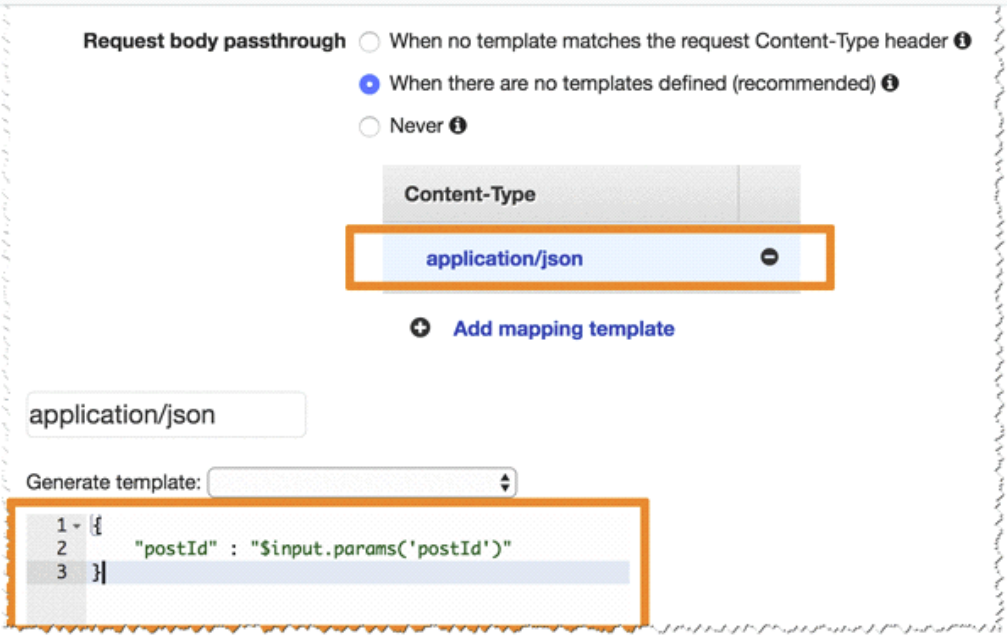

Resources
Actions
/ - GET - Meth
/
GET
POST
METHOD ACTIONS
Edit Method Documentation
Delete Method
RESOURCE ACTIONS
Create Method
Create Resource
Enable CORS
Edit Resource Documentation
API ACTIONS
Deploy API
Import API
Edit API Documentation
Delete API

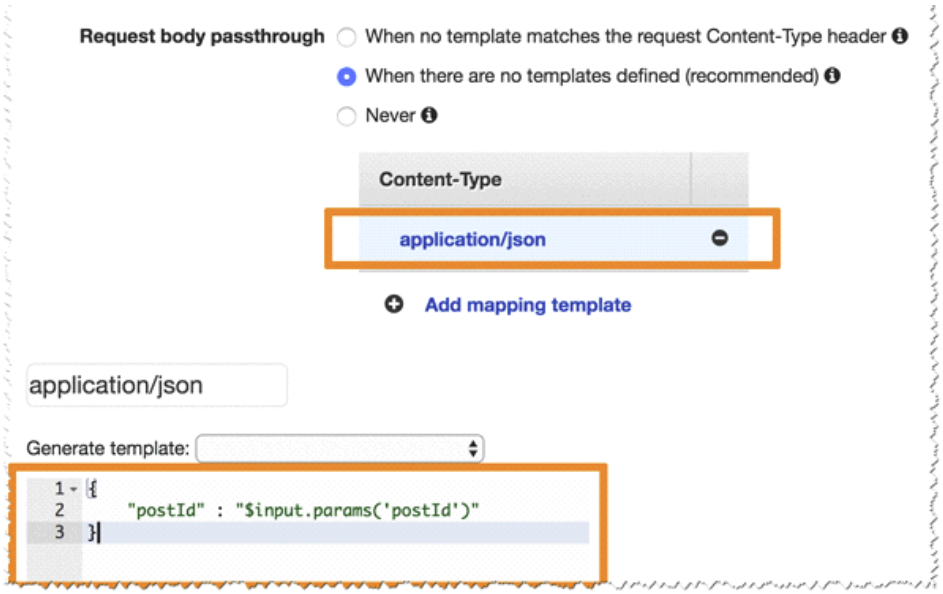


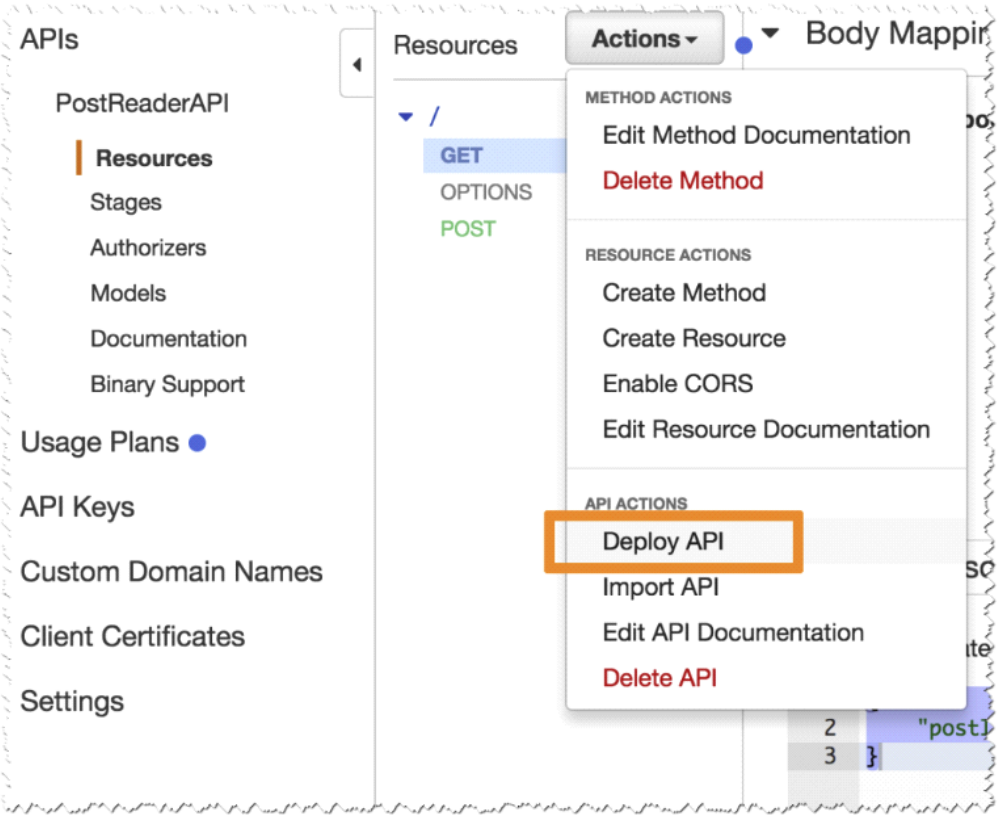
We use the following mapping:

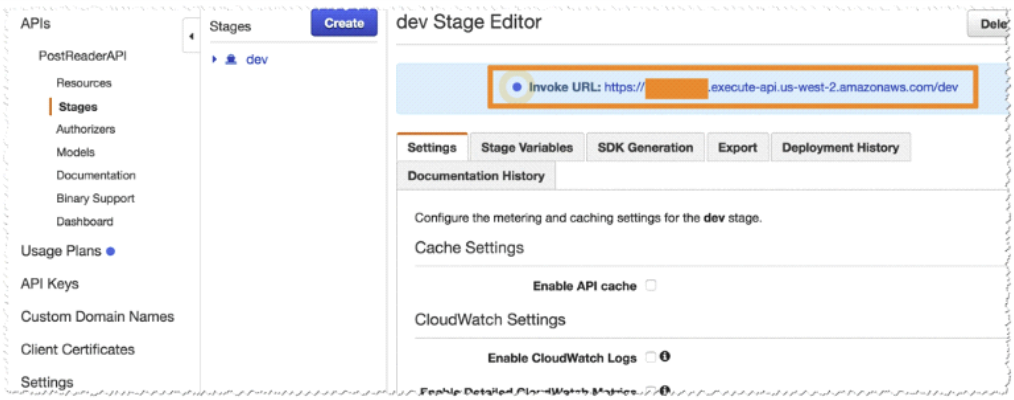
```
{
    "postId" : "$input.params('postId')"
}
```

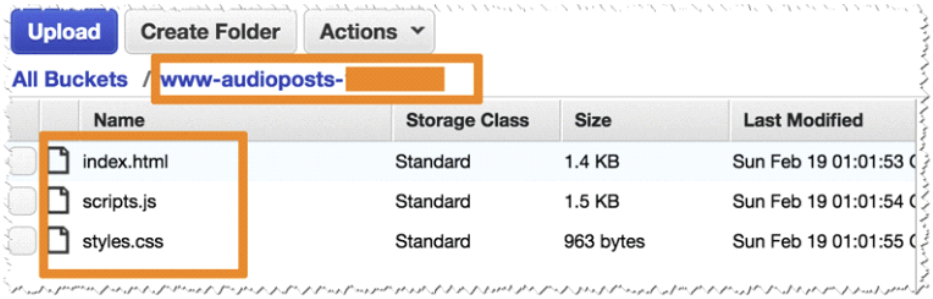




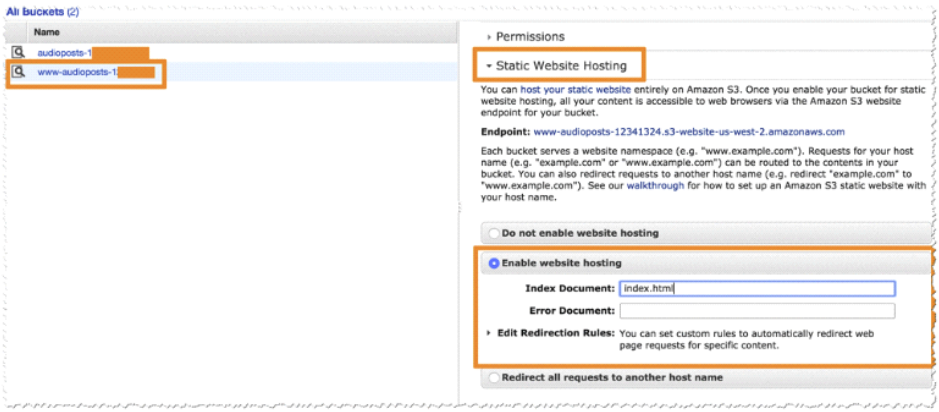

Request body passthrough
When no template matches the request Content-Type header
When there are no templates defined (recommended)
Never
Content-Type
application/json
Add mapping template
application/json
Generate template:
{
"postId" : "$input.params('postId')"
}


APIs
PostReaderAPI
Resources
Stages
Authorizers
Models
Documentation
Binary Support
Usage Plans
API Keys
Custom Domain Names
Client Certificates
Settings
Resources
Actions
Body Mappi
/
GET
OPTIONS
POST
METHOD ACTIONS
Edit Method Documentation
Delete Method
RESOURCE ACTIONS
Create Method
Create Resource
Enable CORS
Edit Resource Documentation
API ACTIONS
Deploy API
Import API
Edit API Documentation
Delete API

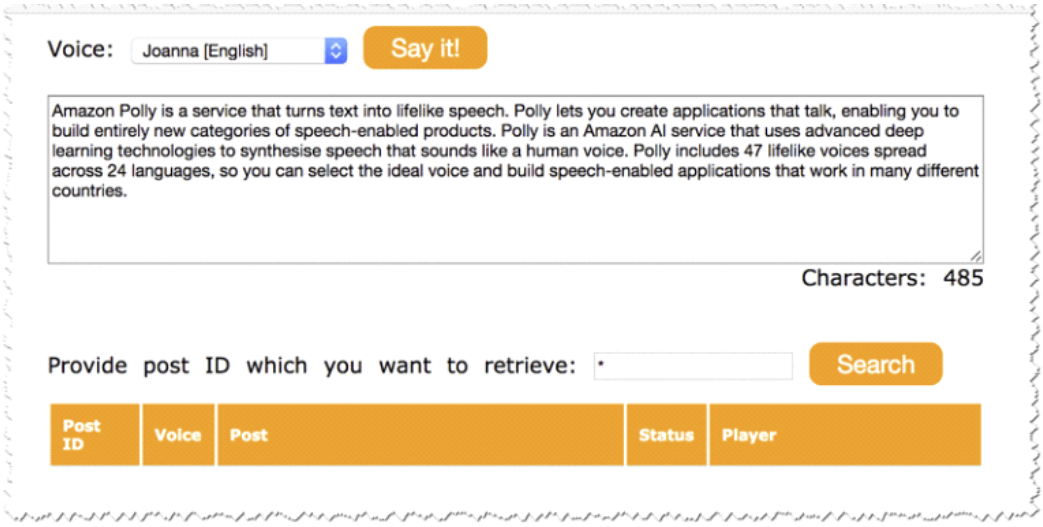


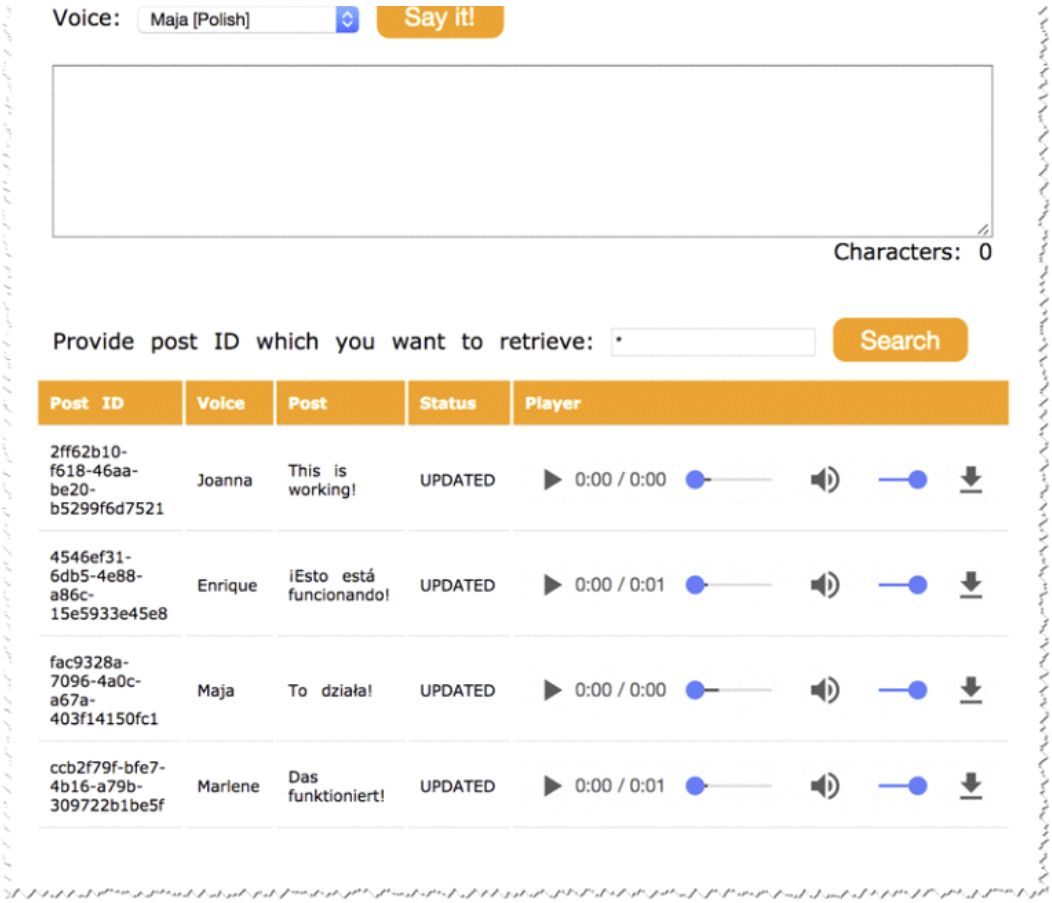


In the properties for the bucket, chose **Static Website Hosting**, enable website hosting, and provide the name of the index file (index.html).




Voice:
Joanna [English]
Say it!
Characters: 0
Provide post ID which you want to retrieve:
Search
Post ID
Voice
Post
Status
Player


Voice:
Joanna [English]
Say it!
Amazon Polly is a service that turns text into lifelike speech. Polly lets you create applications that talk, enabling you to build entirely new categories of speech-enabled products. Polly is an Amazon AI service that uses advanced deep learning technologies to synthesise speech that sounds like a human voice. Polly includes 47 lifelike voices spread across 24 languages, so you can select the ideal voice and build speech-enabled applications that work in many different countries.
Characters: 485
Provide post ID which you want to retrieve:
Search
Post ID
Voice
Post
Status
Player



Conclusion

we created an application that can convert text into speech in dozens of languages and speak that text in even more voices. we can use it for converting text on websites or adding speech functionality in web applications. And we did it completely serverless. There are no servers to maintain or patch, etc. By default, our application is highly available because AWS Lambda, Amazon API Gateway, Amazon S3, and Amazon DynamoDB use multiple Available Zones.

Cloud Computing -

Text Books:

- RajkumarBuyya, J.Broberg, A. Goscinski, “Cloud Computing Principles and Paradigms”, Wiley, 2011.
- A. Srinivasan, J. Surish “ Cloud Computing A Practical Approach for Learning and implementation””, Pearson, 2014.

- Ron Schmelzer et al. “XML and Web Services”, Pearson Education, 2002.
- Thomas Erl, “Service Oriented Architecture: Concepts, Technology, and Design”, Pearson Education, 2005.

References:

- Barrie Sosinsky, “ Cloud Computing Bible”, Wiley., 2010
- Tim Mather, “Cloud Security and Privacy”, O’REILLY., 2009
- https://docs.aws.amazon.com/
- Frank P.Coyle, “XML, Web Services and the Data Revolution”, Pearson Education, 2002
- https://aws.amazon.com/blogs/machine-learning/build-your-own-text-to-speech-applications-with-amazon-polly/