### **Enhancing Applications with AI Text-to-Speech: Using Amazon Polly for Natural Voice Outputs**

Artificial Intelligence (AI) is revolutionizing how we interact with technology, making applications more dynamic, accessible, and user-friendly. One remarkable AI-powered service is Amazon Polly, which converts text into natural-sounding speech. By enabling applications to "speak," Amazon Polly not only improves accessibility for visually impaired users but also adds a new dimension to content consumption by transforming written material into an auditory experience.

This blog explores how Amazon Polly can enhance applications through a practical use case: A News Reader Application Using Amazon Polly and Streamlit. This application converts the text of news articles into speech, making information accessible to users who prefer listening over reading.

### **What is Amazon Polly?**

Amazon Polly is a cloud-based Text-to-Speech (TTS) service offered by AWS. It uses advanced deep learning technologies to synthesize lifelike speech from written text. With Polly, developers can create applications that interact naturally with users, offering a more engaging and inclusive experience.

Some benefits of using Amazon Polly include:

* Natural Voices: Polly supports a variety of voices and languages, ensuring smooth and realistic audio output.
* Scalability: Being a cloud-based service, Polly scales effortlessly to meet your application’s demands.
* Integration: It integrates seamlessly with various programming languages and frameworks.

To demonstrate Polly's capabilities, we will implement a news reader application that extracts content from a news article URL, converts it to speech, and generates an audio file. Below, I provide an overview of how the application works.

#### **Application Workflow overview**

The application is simple yet powerful:

1. A user enters a valid news article URL.
2. The app extracts the article's title and content using Newspaper3k.
3. Amazon Polly converts the text into natural-sounding speech.
4. The user can listen to the generated audio directly on the application or the downloaded .mp3 audio file.

Tools we will be using to build the application are:

* boto3: AWS SDK for Python, which serves as a bridge between Python and AWS services
* streamlit: A Python framework for building interactive web applications.
* Amazon Polly: For generating speech from text.
* newspaper3k: Used for extracting content from news article URLs. While alternative lightweight libraries like ‘beautifulsoup4’ can also extract text from news articles, they may require additional parsing adjustments for certain websites.

**Prerequisites**

Before you begin you need:

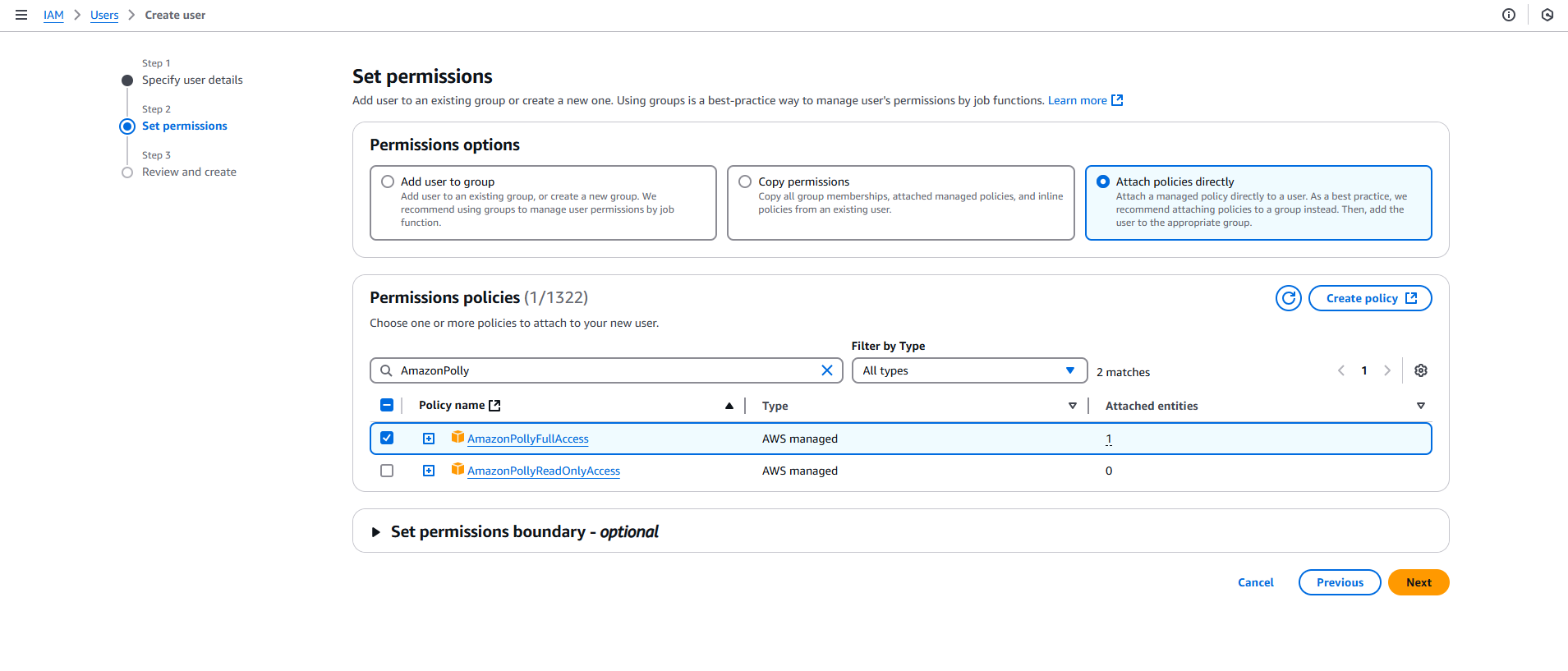
* An aws account with a full administrative privileges
* Python runtime environment
* AWS Command Line Interface (CLI)

**Step 1 : Create an IAM User**

Our Python application will programmatically access Amazon Polly. To achieve this, the application uses an IAM user. In this step we will create that IAM user.

To create an IAM user

1. Navigate to the Identity Access Management (IAM) Console.
2. Go to the Users link in the left side navigation panel
3. Click Create User
4. Provide name for the user and leave the ‘Provide user access to the AWS Management Console’ option unticked.
5. Select ‘Add Permissions’ from the Permissions options.
6. Search and add ‘AmazonPollyFullAccess’ and click next.
7. Review and create the user.



Screenshot of the IAM console’s Add permission page

**Step 2 : Generate Access Key**

To generate an Access Key for the user

1. Go to the security credentials tab, scroll down to the “Access keys” section
2. In the Access keys section, choose Create access key.
3. On the Access key best practices & alternatives page, choose “Command Line Interface (CLI)” tick the “I understand the above recommendation and want to proceed to create an access key.” check box and then click Next.
4. On the Retrieve access keys page, choose either Show to reveal the value of your user's secret access key, or Download .csv file. This is your only opportunity to save your secret access key. After you've saved your secret access key in a secure location, choose Done.

**Step 3 : Initialize Python Application and Install the necessary packages**

Now that we have all aws config setups completed we can now create our python app. This step assumes you have python installed and running.

1. Open Windows Command Prompt
2. CD into you chosen project directory
3. Install the boto3, streamlit, newspaper3k packages

pip install boto3 streamlit newspaper3k

**Note**: The boto3 package is a Python library provided by Amazon Web Services

(AWS) to interact with various AWS services programmatically. It allows you to

manage resources, and integrate AWS services into your Python applications.

1. Install the required dependencies

pip install lxml[html\_clean] lxml\_html\_clean

**Note**: Recently, the ‘lxml.html.clean’ module was split into a separate

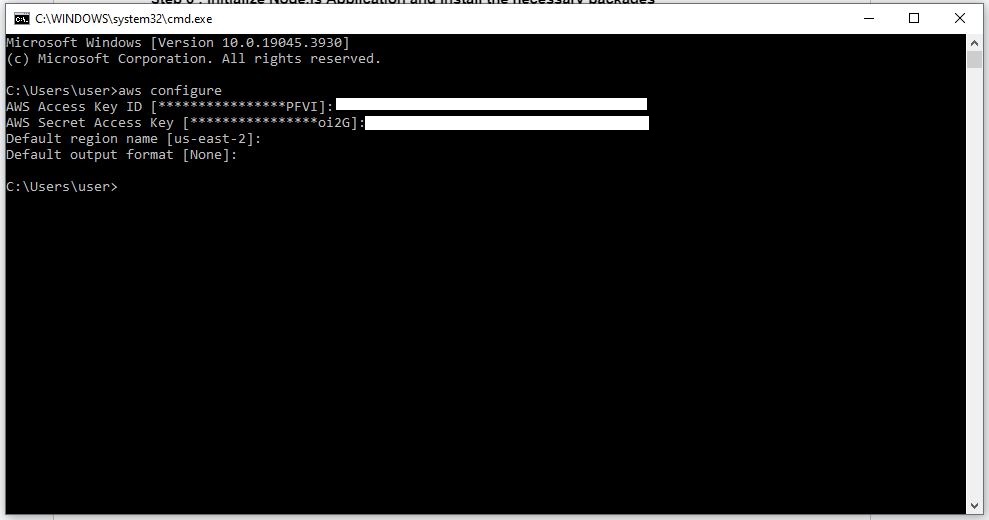
(package) from the ‘lxml’ library. The ‘newspaper3k’ library depends on this

module but hasn't updated its dependencies to handle the change.

**Step 4 : Set up authentication details**

At this point, we are setting up our application to interact with AWS services. To achieve this, we'll be configuring our AWS Command Line Interface (CLI) with the credentials of the user we've already created. Assuming you have the AWS CLI installed, we prefer configuring the user credentials through the CLI rather than hard-coding them into our code. This approach enhances security, as storing sensitive credentials directly in the code is not recommended. Following the AWS credentials provider chain, the AWS SDK will automatically search and use the credential within the AWS CLI, ensuring that our application can securely access AWS bedrock.

1. Initiate the user configuration by entering the command "aws configure."
2. Enter the Access Key ID.
3. Provide the Secret Access Key.
4. Specify the Region.



Screenshot of windows command prompt window showing the process of aws user configuration

### **Step 5: Project Structure Setup**

1. Create a repository named news\_reader\_app. Inside the repository, create a folder called output, along with a requirements.txt file and an app.py file.
2. Add boto3, streamlit, newspaper3k dependencies to requirements.txt file

### **Step 6: Write the Application Code**

Open ‘app.py’ and paste the provided code

import boto3

import streamlit as st

from newspaper import Article

import os

polly = boto3.client("polly")

def synthesize\_speech(text, output\_file):

response = polly.synthesize\_speech(

Text=text,

OutputFormat="mp3",

VoiceId="Joanna" # Choose a voice (Joanna, Matthew, etc.)

)

with open(output\_file, "wb") as file:

file.write(response["AudioStream"].read())

def extract\_article(url):

article = Article(url)

article.download()

article.parse()

return article.title, article.text

st.title("📰 News Reader Application with AI Text-to-Speech")

st.markdown("Enter a valid news article URL below, and the app will convert the article's content to natural speech using Amazon Polly.")

url\_input = st.text\_input("Enter the URL of a news article:")

if st.button("Generate Audio"):

if url\_input:

try:

st.info("Fetching article content, please wait...")

title, article\_text = extract\_article(url\_input)

if len(article\_text.strip()) == 0:

st.error("The article has no content to process. Please try a different URL.")

else:

output\_path = "output/news\_audio.mp3"

os.makedirs("output", exist\_ok=True)

st.info("Synthesizing speech, please wait...")

synthesize\_speech(article\_text[:3000], output\_path)

st.audio(output\_path, format="audio/mp3")

st.success(f"Audio generated successfully! Article Title: {title}")

except Exception as e:

st.error(f"Failed to process the URL. Error: {e}")

else:

st.error("Please enter a valid news article URL.")

**Let's break down the code:**

**Import Required Libraries:**

import boto3

import streamlit as st

from newspaper import Article

import os

Here, the code imports the ‘boto3’ library to interact with AWS services, ‘streamlit’ for the UI, and ‘newspaper3k’ for content extraction and the ‘os’ module which provides a way to interact with the operating system. The ‘os’ module enables you to perform various operating system-related tasks such as accessing environment variables, handling file paths, creating or deleting directories, and managing processes.

**Configuring Amazon Polly:**

**polly = boto3.client("polly")**

A Polly client is created using the AWS SDK (boto3). This allows the app to send text to Polly and receive synthesized speech.

**Synthesizing Speech:**

def synthesize\_speech(text, output\_file):

response = polly.synthesize\_speech(

Text=text,

OutputFormat="mp3",

VoiceId="Joanna" # Choose a voice (Joanna, Matthew, etc.)

)

with open(output\_file, "wb") as file:

file.write(response["AudioStream"].read())

This code defines a function called synthesize\_speech that takes two inputs: some text and a file name for saving the output. When you provide the text, the function sends it to Polly and requests it to be converted into audio using a specific voice, such as "Joanna" or "Matthew." Polly processes the text and sends back the audio as an MP3 file. The function then saves this audio to your computer with the specified file name.

**Extract Article Content:**

**def extract\_article(url):**

**article = Article(url)**

**article.download()**

**article.parse()**

**return article.title, article.text**

The ‘extract\_article’ function is a key component of the application, responsible for retrieving the title and main body text of a news article from a given URL. It leverages the powerful Newspaper3k library, which is specifically designed for web scraping and article parsing. Here’s how it works step by step:

‘**Article(url)**’ initializes an object that represents the web page at the specified URL. This object will later store the extracted information.

**‘article.download()’** method fetches the raw HTML content of the web page. This step is analogous to retrieving the source code of the webpage.

**‘article.parse()’** analyzes the structure of the web page to extract meaningful content such as the article title and text. Newspaper3k is optimized for handling the complex structure of news websites, allowing it to distinguish between main content, navigation menus, advertisements, and other elements.

The ‘**article.title’** and **‘article.text’** properties store the extracted data. The title is the headline of the article, while the text contains the main body content. These are returned as a tuple to be used in subsequent steps of the application.

**Title and Input Field:**

### **st.title("📰 News Reader Application with AI Text-to-Speech")**

### **st.markdown(**

### **"Enter a valid news article URL below, and the app will convert the article's content to natural speech using Amazon Polly."**

### **)**

### **url\_input = st.text\_input("Enter the URL of a news article:")**

### The ‘st.title’ function sets the title of the application to "📰 News Reader Application with AI Text-to-Speech," displayed prominently at the top of the page. The ‘st.text\_input’ function creates a text input box where users can enter the URL of the news article they want to convert to speech. The user input is stored in the variable url\_input. This step establishes a simple and intuitive interface for users to interact with the application.

**Audio generation from News URL:**

**if st.button("Generate Audio"):**

**if url\_input:**

**try:**

**st.info("Fetching article content, please wait...")**

**title, article\_text = extract\_article(url\_input)**

**if len(article\_text.strip()) == 0:**

**st.error("The article has no content to process. Please try a different URL.")**

**else:**

**output\_path = "output/news\_audio.mp3"**

**os.makedirs("output", exist\_ok=True)**

**st.info("Synthesizing speech, please wait...")**

**synthesize\_speech(article\_text[:3000], output\_path)**

**st.audio(output\_path, format="audio/mp3")**

**st.success(f"Audio generated successfully! Article Title: {title}")**

**except Exception as e:**

**st.error(f"Failed to process the URL. Error: {e}")**

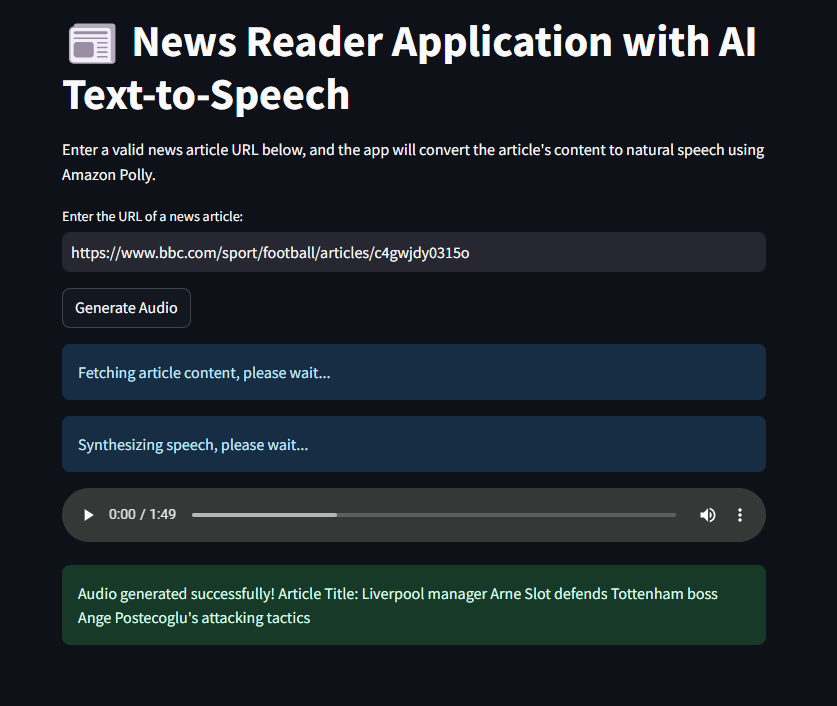
**else:**

**st.error("Please enter a valid news article URL.")**

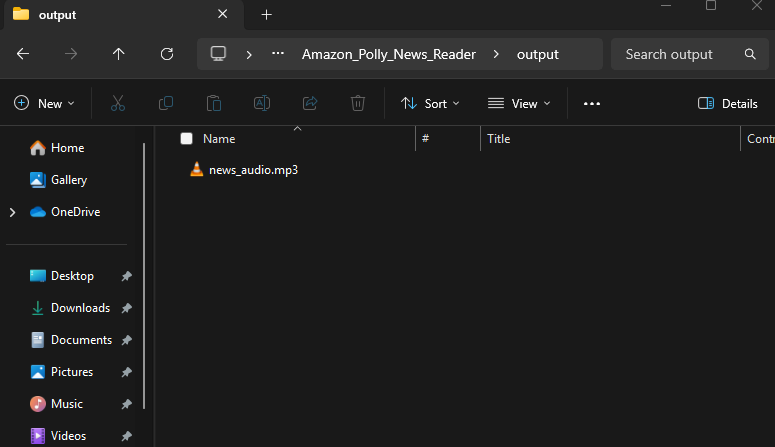
This code snippet is the core logic of the application. It handles the "Generate Audio" button click, fetches the article content, converts it to speech, and plays the audio. Let’s break down the key parts of this functionality:

1. Button Interaction:
   * The st.button("Generate Audio") checks if the button is clicked.
   * Inside, it ensures that a URL has been provided using the if url\_input condition. If no URL is provided, an error message prompts the user to enter a valid one.
2. Fetching Article Content:
   * When a URL is entered, the app displays an informational message, "Fetching article content, please wait...".
   * The extract\_article(url\_input) function is called to retrieve the article title and text.
   * If the extracted article text is empty (if len(article\_text.strip()) == 0), an error message alerts the user to try a different URL.
3. Generating and Saving Audio:
   * The app creates an output directory using os.makedirs to store the synthesized audio file (news\_audio.mp3).
   * It passes up to 3000 characters of the article content to the synthesize\_speech function for processing.
   * Another informational message, "Synthesizing speech, please wait...", keeps the user informed during this operation.
4. Playing the Audio:
   * After successful synthesis, st.audio is used to play the audio directly in the app, and a success message displays the article title.
5. Error Handling:
   * A try-except block ensures graceful handling of errors, such as issues with the URL or the synthesis process. If an error occurs, it is displayed to the user.
   * If no URL is provided, a specific error message is shown: "Please enter a valid news article URL."

**Outputs:**



Screenshot of the final application

****

Screenshot of the generated audio file in the output folder

### 

### **Conclusion**

Amazon Polly empowers developers to make applications more engaging and inclusive by enabling text-to-speech capabilities. Whether for accessibility, convenience, or innovation, Polly's natural voice outputs can transform how users interact with applications.

The news reader application demonstrates just one use case, but the possibilities are vast. By integrating Polly into your projects, you can take user experience to the next level. So, why not start today and give your applications a voice?