**Creating a Text-to-Speech App with Azure: A Writer’s Guide**

I am a writer who likes to envision my fictional scenes like a movie, and write them as I see and hear them. Novelists, scriptwriters, and game writers all use this technique from time to time. This ensures that the characters all have distinctive voices and the dialogue flows more naturally–a common writing pitfall.

**I am also a Python coder training in Azure cloud architectures**. A text-to-speech app written as a function sounded like a useful way for me to learn new skills, and language is in my wheelhouse. It is also easy enough that even someone with minimal coding experience should be able to complete the project (not all writers code).

**I decided to write an app specifically for writers to hear the voices of their characters.** Azure provides robust AI-based tools to make this project a reality. I was excited to dig in and create a tool I can actually use! Text-to-Speech is also helpful in making apps like chatbots, but I’m not getting that complex here.

**Starting Point: Using CoPilot for Initial Guidance**

First, I went to CoPilot, Microsoft Edge’s AI helper, as a starting point. AI on the internet is not perfect. For coding purposes, it often uses old articles, libraries, or tools that don’t work anymore. I then went to the articles CoPilot referenced, and read those. For those of you new to coding, the first rule of coding is: *Read the documentation*. (The second is: *Use clear, concise comments so everyone knows what your code does*.)

The articles CoPilot referenced were both from Microsoft and therefore up-to-date. The first one was a text-to-speech quickstart guide and the second Azure’s AI speech documentation. This was helpful at getting me on the right path. Simply type in “Azure text-to-speech quickstart guide” and “Azure AI speech documentation” to find the proper documentation.

**Understanding Amazon Text-To-Speech Pricing**

**Be careful to check the pricing before starting this or any application using any service**. It is a pay-as-you-go model, so you’re only charged for what you consume. According to Microsoft, you are billed for each character turned into speech–that is, an A.I. voice reading your words.. For Chinese characters, Japanese kanji, and Korean hanja, you are billed for two characters. You get 0.5 million characters free per month for Text to Speech, so that was a relief when I started this project.

If I wanted entire novels read aloud, I would probably begin by putting my novel in blob storage, then having the text-to-speech app read it from there. For most novels, that would probably take hours to hear, but would be helpful if you are proofreading your novel before it goes to print. The human eye often skips typos. The speech reader will read grammar mistakes aloud so you can catch them. This project does not cover reading entire novels, but it is possible.

### **Before You Begin: Creating an Azure Account**

### If you don’t have one, you can create a free Azure account. Generally speaking, the free account is for a $200 credit for 30 days, and many services free for 1 year. Certain services are not free or might not be available with a free account. Use the Microsoft documentation to create your account, or watch a recent video or one directly from Microsoft on YouTube. I already set up my Azure Account, so I was ready to log in. Just go to [portal.azure.com](http://portal.azure.com) and follow the steps to log in.

**Coding the Text-to-Speech App**

**Step 1: Creating a Speech Resource**

Go to the Azure portal and create a Speech resource. This will give you access to the necessary keys and endpoints. I logged in and went to my new best friend, the Azure search bar. I typed in “speech service” and began.

### 

### **Step 2: Installing the Speech SDK**

I used the code editor VSCode to write my application. You can use whatever text editor you like. VSCode has an Azure plugin so I can directly connect to Azure.

In order to use Azure’s Speech SDK to interact with Azure’s Text-To-Speech (TTS) services, I had to install it. The SDK is available for various programming languages, including Python, C#, and JavaScript. I use Python, so I put this in the command line of my code editor:

**pip install azure-cognitiveservices-speech**

Text--to-speech is also available with the REST API and the Speech CLI. I used the Speech SDK for this project

### **Step 3: Authenticating Your Application**

Set up environment variables for your Speech resource key and region:

export SPEECH\_KEY=your-key

export SPEECH\_REGION=your-region

**Highly Important Note:** *Keep your keys safe.* Don’t put that information where anyone can get it–and run up charges on your account. You need to save this in your .env file if you are testing your code in a text editor. Or, Azure can keep your secrets safe in its Key Vault. The vault is not free.

**Creating an .env file:**

Note that below I have the variables “AZURE\_SPEECH\_KEY” and “AZURE\_SERVICE\_REGION.” Create a file called .env and put both variables in with an = sign.

Use pip install python-dotenv if you are using Python. See the below Python file to see how to call these variables.

You can also save and use keys in the Microsoft GUI, but I didn’t do that for this project.

### **Step 4: Write Your TTS Application: Text to Speech Only**

I started in VSCode by clicking on Azure on the side, then clicking Local Function, then right clicking Local Function. Then I clicked Create Function. This created files like a local.settings.json file. You also need a .venv file.

Of course, you probably don’t have to do this on your IDE. I had VSCode create the .json and .venv files for me.

* local.settings.json

{

*"IsEncrypted"*: **false**,

*"Values"*: {

*"AzureWebJobsStorage"*: "UseDevelopmentStorage=true",

*"FUNCTIONS\_WORKER\_RUNTIME"*: "python",

*"AZURE\_SPEECH\_KEY"*: "YourSubscriptionKey",

*"AZURE\_SERVICE\_REGION"*: "YourServiceRegion"

}

}

* Host.json:

{

*"version"*: "2.0",

*"logging"*: {

*"applicationInsights"*: {

*"samplingSettings"*: {

*"isEnabled"*: **true**,

*"maxTelemetryItemsPerSecond"*: 20,

*"excludedTypes"*: "Request;Exception"

}

}

},

*"extensionBundle"*: {

*"id"*: "Microsoft.Azure.Functions.ExtensionBundle",

*"version"*: "[4.\*, 5.0.0)"

}

}

My keys are in the .env file instead, NOT the local settings file. I have a healthy fear of my keys getting out and getting into trouble without me.

**Making the .py file:**

Here’s a basic example in Python to convert text to speech:

import azure*.cognitiveservices.speech* as speechsdk

import os

from dotenv import load\_dotenv

load\_dotenv()

speech\_key = os*.getenv("AZURE\_SPEECH\_KEY")*

service\_region = os*.getenv("AZURE\_SERVICE\_REGION")*

speech\_config = speechsdk*.SpeechConfig(subscription=speech\_key, region=service\_region)*

synthesizer = speechsdk*.SpeechSynthesizer(speech\_config=speech\_config)*

ssml\_string = *f*"""

<speak version='1.0' xmlns='http://www.w3.org/2001/10/synthesis' xml:lang='en-US'>

<voice name='en-IE-ConnorNeural'>

Hello, this is a test for Azure Text-to-Speech.

</voice>

</speak>

"""

result = synthesizer*.speak\_ssml\_async(ssml\_string).get()*

if result*.reason* == speechsdk*.ResultReason.SynthesizingAudioCompleted*:

with open("local\_output1.wav", "wb") as audio\_file:

audio\_file*.write(result.audio\_data)*

else:

print(*f*"Error: *{*result*.reason}*")

As you can see, my speech key and service region are safely hidden in my .env file.

**Saving the File:**

Save this as text\_to\_speech.py, or maybe the tried-and-true main.py.

**Saving the Output:**

This file will save into the same folder as your app. If you want it saved somewhere else, you’ll have to specify the path to that folder. This code will write over the previous file with the same name. **Before you run it** you could save the output with a new file name, such as adding a 1 or, if you changed the Neural voice name, the name of the voice, such as Ryan1\_local\_output.wav. You will have to change the file name next to *with open* to save it as a new file name. Do this if you want to save the sounds of different voices.

### **Step 5: Testing and Deployment**

**Test Locally**: Run your application locally to ensure it works as expected.

You can run it in the debugger, such as in VSCode. Just open the Command Palette and select “Azure Functions: Start Debugging.” Or simply run it. You can also type “func start” into the terminal.

### **Optional: Step 6: Deploying to Azure**

You can deploy your application to Azure using services like Azure App Service or Azure Functions for a scalable solution. You do not have to do this; but moving it online ensures even if something goes wrong locally that it will still run. Remember, you are charged for using a function. **Remember that Azure’s free service credit of $200 expires after thirty days.**

**Using VSCode**

In VSCode, open the command palette with F1. At the prompt, enter, then select **Azure Functions: Create Function App in Azure**. Follow the prompts to select your subscription, enter a **globally unique** name for the function app, select a runtime stack, and select a location for your resources.

Then go back to the Command Palette, enter, and select Azure Functions: Deploy to Function App. Select Deploy. Use View Output to view the results of the creation and deployment. Use this article to help you if you get confused or stuck: [Create a Python function using Visual Studio Code - Azure Functions | Microsoft Learn](https://learn.microsoft.com/en-us/azure/azure-functions/create-first-function-vs-code-python).

**Using Azure**

Go into Azure Functions. Click the Add+, and follow the instructions to add your function. *Be sure to add it to your resource group.* If you ever want to start over completely, you can delete the entire resource group.

**Step 7: Troubleshooting**

* If you can’t hear your file, check the Speech Service quotas. There are limits or quotas on your Azure Speech subscription.
* If you still can’t hear your .wav file, see the Text to Speech .wav Only option below in the Advanced Tips section.
* If you get stuck at any point, use a search engine to search for an article to help. You can watch a YouTube video on deploying Azure functions. You can also use CoPilot, ChatGPT, or other AI assistants to help you if you make mistakes. I definitely used both CoPilot and ChatGPT to help find my error when I couldn’t hear my audio.

**Advanced Tips: Changing the Code**

**SSML**

Note the SSML code in the middle of the above file. Speech Synthesis Markup Language (SSML) is an XML-based markup language for speech.

**Prosody and Speed**

Prosody is the rhythm and tones of speech, which add emotional depth. Speed is how fast someone speaks. In the above file, I did NOT use prosody to change the speed and rhythm of the voice…but you can. Manipulate the prosody rate and pitch as you see fit. Simply replace:

ssml\_string = *f*"""

<speak version='1.0' xmlns='http://www.w3.org/2001/10/synthesis' xml:lang='en-US'>

<voice name='en-IE-ConnorNeural'>

Hello, this is a test for Azure Text-to-Speech.

</voice>

</speak>

"""

with:

ssml\_string = *f*"""

<speak version='1.0' xmlns='http://www.w3.org/2001/10/synthesis' xml:lang='en-US'>

<voice name='en-IE-RyanNeural'>

<break time='1s' />

<prosody rate='-10%' pitch='-5%'>

Hello, this is a test for Azure Text-to-Speech.

</prosody>

</voice>

</speak>

"""

….and change the percentages to manipulate the voice.

**Changing The Neural Voice**

Notice that I changed the name of the Neural voice from ConnorNeural to RyanNeural. You can do that in the first one, too. You can play with the voices to find the one that sounds like the voice for your character you have in your head.

**BalancingTime and Money**

Playing with the voice is extremely fun, but the point is to enhance your writing, not perfect digital voices. I suggest only changing the voice, prosody, and/or pitch (and maybe even emotion) when writing a new character. This article has some examples of speech and emotion:

[Announcing new voices and emotions to Azure Neural Text to Speech | Microsoft Azure Blog](https://azure.microsoft.com/en-us/blog/announcing-new-voices-and-emotions-to-azure-neural-text-to-speech/?msockid=12bfb702810f6b152933a4e480816aff).

I wrote my code so it can be tweaked to change the voices, but give yourself a certain amount of time on a timer to change the voice to prevent the “rabbit hole phenomenon.” Only use a sentence or two to hear the voice.

### **Writing Your Text-to-Speech Application: Text to Speech .wav Only**

You might not want to hear your file, and want to save it as a .wav file only. The weird “base 64” import is so the .wav file is encoded properly. Here’s a basic example in Python to convert text to speech and save it as a .wav file:

import requests

import base64

# Replace with the correct URL for your Azure Function endpoint

url = "http://localhost:7071/api/TextToSpeech"

data = {

"text": "Hello, this is a test for Azure Text-to-Speech.",

"voice": "en-IE-ConnorNeural"

}

# Send the POST request to the Azure Function

response = requests*.post(url, json=data)*

if response*.status\_code* == 200:

# Decode the Base64 response and save it as a .wav file

audio\_data = base64*.b64decode(response.text)*

with open("output.wav", "wb") as file:

file*.write(audio\_data)*

print("Audio saved as output.wav")

else:

print(*f*"Request failed with status *{*response*.status\_code}*: *{*response*.text}*")

This file will save into the same folder as your app. If you want it saved somewhere else, you’ll have to specify the path to that folder.

**Changing the Neural Voice**

Like the previous file, you can change the Neural voice. Unless you want to write over the same file over and over, you can go in and change the name of the .wav file with a number, or the name of the voice you’re using (such as Ryan1\_output.wav).

**Switching Python Files**

If you want to switch between the voice and the only .wav Python files, be sure to close the terminal in between. If your local machine throws up an error about using the same port, change the port number by 1 (like 7071 to 7072) in the URL. You can also do func init, then func start to be sure your file starts out fresh.

**Conclusion**

This was a fun project for me to do, and is useful for when I’m writing, not coding. I just have to be sure:

1. I stay within the subscription and my spending limits and
2. I refrain from falling down the rabbit hole of trying out voices, prosody, and emotions to the point of not writing (or coding).

Happy coding!

### **Explore the Code**

The complete code for this project is available on my GitHub: <https://github.com/lauralstephenson/azure_text_to_speech_project_final/tree/master>

Feel free to explore, fork, or contribute!

**References**

“Azure AI Speech Pricing” (2024) *Microsoft Azure.* [Azure AI Speech Pricing | Microsoft Azure](https://azure.microsoft.com/en-us/pricing/details/cognitive-services/speech-services/)

Beatman, Andy. (2022, May 9). Announcing new voices and emotions to Azure Neural Text to Speech. [web log comment] Retrieved from [Announcing new voices and emotions to Azure Neural Text to Speech | Microsoft Azure Blog](https://azure.microsoft.com/en-us/blog/announcing-new-voices-and-emotions-to-azure-neural-text-to-speech/?msockid=12bfb702810f6b152933a4e480816aff)

Santoso, Danoe (2024, August 21). Microsoft Azure Text to Speech Pricing: Models, Factors Affecting, & Cost Optimization. *Octobits*. [Microsoft Azure Text to Speech Pricing: Models, Factors Affecting, & Cost Optimization - Octobits Learning Center](https://blog.octobits.io/digital-transformation/microsoft-azure-text-to-speech-pricing/#:~:text=Azure%20Text%20to%20Speech%20is%20a%20pay-as-you-go%20service.,audio%20fidelity%2C%20the%20Neural%20HD%20voice%20is%20available.)

“Quickstart: Create a function in Azure with Python using Visual Studio Code” (2024, September 10). *Microsoft Learn.* [Create a Python function using Visual Studio Code - Azure Functions | Microsoft Learn](https://learn.microsoft.com/en-us/azure/azure-functions/create-first-function-vs-code-python)

Medium article link: <https://medium.com/@lstephenson556/creating-a-text-to-speech-app-with-azure-a-writers-guide-46cbf42556f0>