Cecci Riccardo
Matr: 20023915

It Speaks! Create Synthetic Speech Using Text-to-Speech

Objectives

The goal of this project is, through the Text-to-Speech API provided by google cloud platform, to create audio files that replicate human speech based on text or SSML files (SSML stands for Speech Synthesis Markup Language and we will discuss it below).

SSML

SSML is part of a larger set of markup specifications for voice browsers. It is designed to provide a rich, XML-based markup language for assisting the generation of synthetic speech in Web and other applications. The essential role of the markup language is to give authors of synthesizable content a standard way to control aspects of speech output such as pronunciation, volume, pitch, rate, etc. across different synthesis-capable platforms.

A Text-To-Speech system that supports SSML will be responsible for rendering a document as spoken output and for using the information contained in the markup to render the document as intended by the author.

SSML is important because it allows you to control many details of speech, such as emphasis, pitch or speed.

SSML defines an XML format with the <speak> element as its root. Each of the elements present in the XML representation of the input affects the output of the synthetic speech.

The following is an example of SSML markupt:

```
<speak>
  Here are <say-as interpret-as="characters">SSML</say-as> samples.
  I can pause <break time="3s"/>.
  I can play a sound
  <audio src="https://www.example.com/MY_MP3_FILE.mp3">didn't get your
MP3 audio file</audio>.
  I can speak in cardinals. Your number is <say-as interpret-
as="cardinal">10</say-as>.
  Or I can speak in ordinals. You are <say-as interpret-
as="ordinal">10</say-as> in line.
  Or I can even speak in digits. The digits for ten are <say-as
interpret-as="characters">10</say-as>.
  I can also substitute phrases, like the <sub alias="World Wide Web
Consortium">W3C</sub>.
  Finally, I can speak a paragraph with two sentences.
  <p><s>This is sentence one.</s><s>This is sentence two.</s></p>
</speak>
```

Instead, the following example shows how to format the SSML input included in a JSON object.

```
"{
    'input':{
     'ssml':'<speak>The <say-as interpret-as=\"characters\">SSML</say-
as>
          standard <break time=\"1s\"/>is defined by the
          <sub alias=\"World Wide Web Consortium\">W3C</sub>.</speak>'
    },
    'voice':{
      'languageCode':'en-us',
      'name':'en-US-Standard-B',
      'ssmlGender':'MALE'
    },
    'audioConfig':{
      'audioEncoding':'MP3'
    }
  }"
```

You can find more information at:

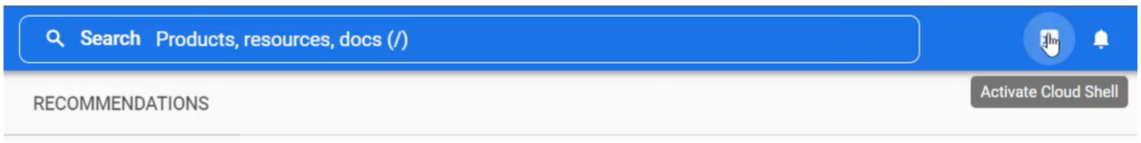
https://www.w3.org/TR/speech-synthesis/

https://cloud.google.com/text-to-speech/docs/ssml

Task #0 : Activate Cloud Shell

Cloud Shell is a virtual machine that is loaded with development tools and provides command-line access to your Google Cloud resources.

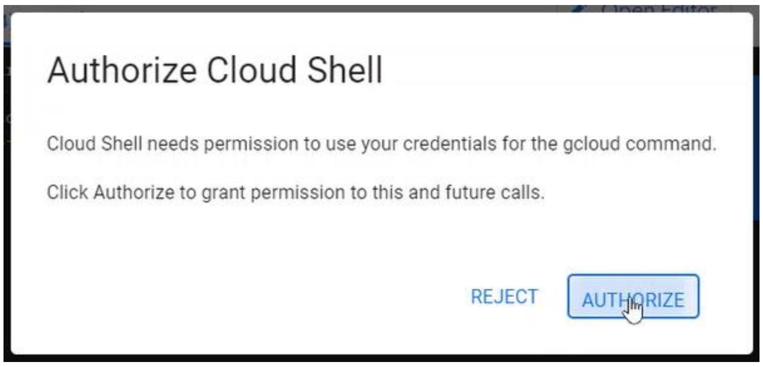
You can activate the cloud shell in the upper right-hand corner of your screen near the search bar.



You can list the active account name with this command:

```
gcloud auth list
```

Then you have to click authorize:


Authorize Cloud Shell

Cloud Shell needs permission to use your credentials for the gcloud command.

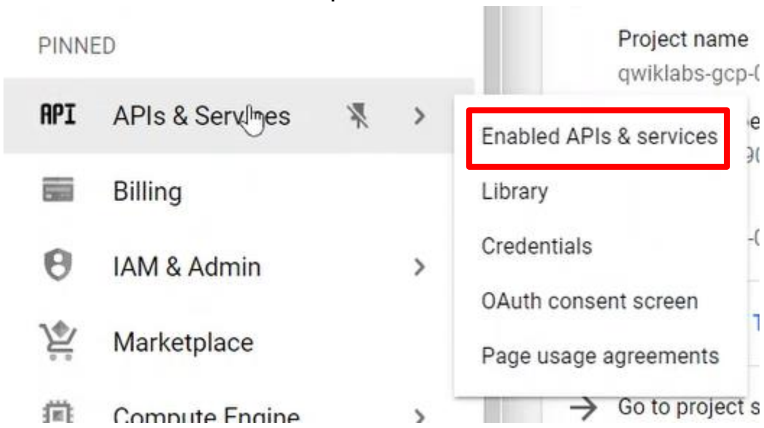
Click Authorize to grant permission to this and future calls.

You can also list the project ID with this command:

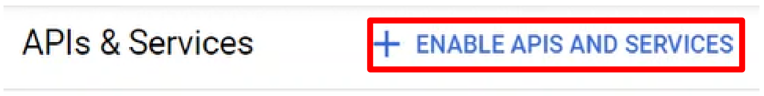
```
gcloud config list project
```

Task #1 : Enable the Text-to-Speech API

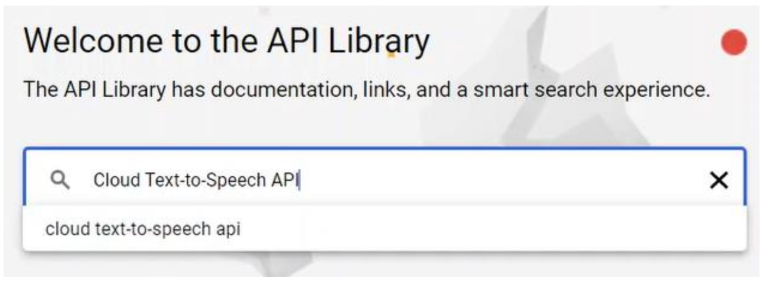
Via the **APIs and Services** menu at the top left, select **Enabled APIs and Services**.



On the page that opens, select **Enable APIs and Services** just below the search bar



and then search for **Cloud Text-to-Speech API.**

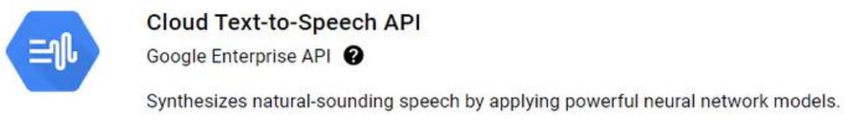

Welcome to the API Library

The API Library has documentation, links, and a smart search experience.

Cloud Text-to-Speech API

cloud text-to-speech api

You will now have several APIs to choose from. Select and enable the API **Cloud Text-to-Speech API** with the hexagonal icon that has the following description: *Synthesizes natural-sounding speech by applying powerful neural network models*


Cloud Text-to-Speech API

Google Enterprise API

Synthesizes natural-sounding speech by applying powerful neural network models.

Wait for a few seconds for the API to be enabled for the project.

Task #2 : Create a virtual environment

Python virtual environments are used to isolate package installation from the system.
First of all, you have to install the **virtualenv** environment with the following command:

```
apt-get install -y virtualenv
```

You may be asked to confirm the installation. If [Y/n] is printed, press y and then enter.
You may now get Permission denied error, if this happens run the command

```
sudo apt-get install -y virtualenv
```

Now you have to build the virtual environment with this command

```
python3 -m venv venv
```

And then activate it

```
source venv/bin/activate
```

Task #3 : Create a service account

You must use a service account to authenticate calls to the Text-to-Speech API.
A service account is a particular type of account used by an application or computing workload, such as a virtual machine (VM) Compute Engine instance, rather than a person. Applications use service accounts to make authorised API calls, authorised as service accounts themselves or as Google Workspace or Cloud Identity users via domain-level delegation.

First of all, you need to create a service account, so run the following command in the Cloud-Shell:

```
gcloud iam service-accounts create tts-qwiklab
```

Now you need the name of your Google cloud project. You can retrieve it from the home page where you started the lab, under the *End Lab* button (there you will find all the credentials), or you can run this command and copy the output:

```
gcloud config get-value project 2> /dev/null
```

Now you need to generate a key to use that service account.
To create and download a key, run the following command in Cloud Shell, replacing *[PROJECT_ID]* with the ID of your Google Cloud project that you copied before (square brackets must be removed when entering the project ID).

```
gcloud iam service-accounts keys create tts-qwiklab.json --iam-account
tts-qwiklab@[PROJECT_ID].iam.gserviceaccount.com
```

Finally, set the GOOGLE_APPLICATION_CREDENTIALS environment variable to the location of your key file:

```
export GOOGLE_APPLICATION_CREDENTIALS=tts-qwiklab.json
```

Task #4 : Get a list of available voices

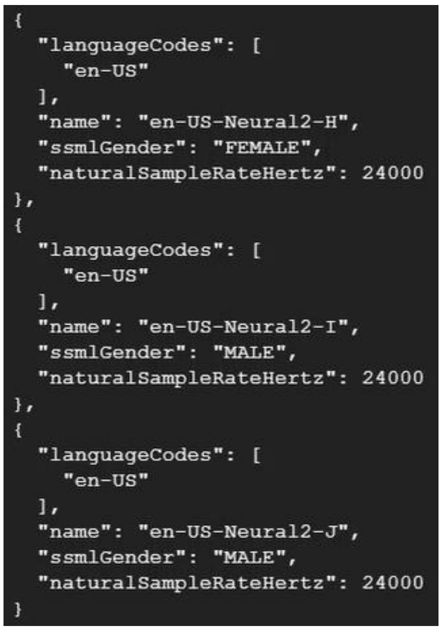
The Text-to-Speech API provides many different voices and languages that you can use to create audio files. There are 3 types of voices

- Standard
- WaveNet
- Neural2

To get the list of all voices available you have to run this *curl* command:

```
curl -H "Authorization: Bearer "$(gcloud auth application-default print-access-token) \
      -H "Content-Type: application/json; charset=utf-8" \
      "https://texttospeech.googleapis.com/v1/voices"
```

the output should be a JSON-formatted result similar to this



```
{
  "languageCodes": [
    "en-US"
  ],
  "name": "en-US-Neural2-H",
  "ssmlGender": "FEMALE",
  "naturalSampleRateHertz": 24000
},
{
  "languageCodes": [
    "en-US"
  ],
  "name": "en-US-Neural2-I",
  "ssmlGender": "MALE",
  "naturalSampleRateHertz": 24000
},
{
  "languageCodes": [
    "en-US"
  ],
  "name": "en-US-Neural2-J",
  "ssmlGender": "MALE",
  "naturalSampleRateHertz": 24000
}
```

Note: In the image there are only 3 voices, but your output will be much longer and will contain several voices in different languages. Each voice field has the same structure:

- *languageCodes* → language codes associated with that voice.
- *name* → the ID of the voice that you provide when you request that voice.
- *ssmlGender* → the gender of the voice to speak the text
- *naturalSampleRateHertz* → The sampling rate of the voice.

You can limit the voices displayed by specifying a single languageCode

```
curl -H "Authorization: Bearer "$(gcloud auth application-default print-
access-token) \
      -H "Content-Type: application/json; charset=utf-8" \
      "https://texttospeech.googleapis.com/v1/voices?language_code=en"
```

Task #5 : Create synthetic speech from text

Now it is necessary to construct the request to the Text-to-Speech API with a file named *synthesize-text.json*

```
touch synthesize-text.json
```

Now you have to edit this file, use *nano* (or the line editor you prefer)

```
nano synthesize-text.json
```

And write something with this structure:

```
{
      'input':{
      'text':'here goes the text you want the synthesiser to reproduce'
      },
      'voice':{
      'languageCode':'en-gb',
      'name':'en-GB-Standard-A',
      'ssmlGender':'FEMALE'
      },
      'audioConfig':{
      'audioEncoding':'MP3'
      }
}
```

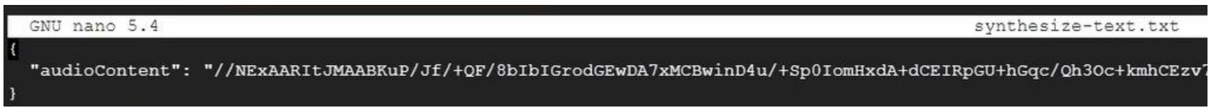
As you can see:

- *input* : text to translate in synthetic speech
- *voice* : the voice you want to use, with all its parameters discussed above
- *audioConfig* : what audio encoding to produce

Now you have to call the Text-to-Speech API, so run this command:

```
curl -H "Authorization: Bearer "$(gcloud auth application-default print-
access-token) \
  -H "Content-Type: application/json; charset=utf-8" \
  -d @synthesize-text.json
"https://texttospeech.googleapis.com/v1/text:synthesize" \
  > synthesize-text.txt
```

This command will call the Text-to-Speech API and it will produce a file called *synthesize-text.txt.* If you open this file, you will notice a particular data in it, as shown in the following image.



```
  GNU nano 5.4                                                                         synthesize-text.txt
{
  "audioContent": "//NExAARItJMAABKuP/Jf/+QF/8bIbIGrodGEwDA7xMCBwinD4u/+Sp0IomHxdA+dCEIRpGU+hGqc/Qh3Oc+kmhCEzv
}
```

This is because the Text-to-Speech API provides the audio output in base64-encoded text assigned to the audioContent field. So now you have to translate this into audio, you need to select the audio data it contains and decode it into an audio file. There are many ways that you can do this, in this lab you'll use some Python code.

Create a file named *tts_decode.py* and add the following code:

```python
import argparse
from base64 import decodebytes
import json
"""
Usage:
      python tts_decode.py --input "synthesize-text.txt" \
      --output "synthesize-text-audio.mp3"
"""
def decode_tts_output(input_file, output_file):
      """ Decode output from Cloud Text-to-Speech.
      input_file: the response from Cloud Text-to-Speech
      output_file: the name of the audio file to create
      """
      with open(input_file) as input:
      response = json.load(input)
      audio_data = response['audioContent']
      with open(output_file, "wb") as new_file:
            new_file.write(decodebytes(audio_data.encode('utf-8')))
if __name__ == '__main__':
      parser = argparse.ArgumentParser(
      description="Decode output from Cloud Text-to-Speech",
      formatter_class=argparse.RawDescriptionHelpFormatter)
      parser.add_argument('--input',
                        help='The response from the Text-to-Speech API.',
                        required=True)
      parser.add_argument('--output',
                        help='The name of the audio file to create',
                        required=True)
      args = parser.parse_args()
      decode_tts_output(args.input, args.output)
```

Now, to create an audio file from the response you received from the Text-to-Speech API, run the following command from Cloud Shell:

```
python tts_decode.py --input "synthesize-text.txt" --output "synthesize-
text-audio.mp3"
```

This will create an MP3 file named *synthesize-text-audio.mp3*. But you cannot play this audio from the Cloud Shell, so you have to create a web server hosting a simple web page that allows you to reproduce this MP3.

Then, create a file *index.html* (touch index.html) and then edit it (nano index.html), with this structure:

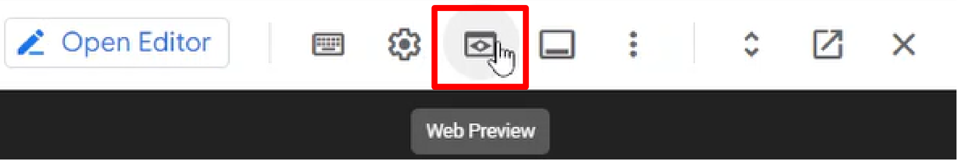
```
<html>
  <body>
  <h1>Cloud Text-to-Speech codelab</h1>
  <p>
  Output from synthesizing text:
  </p>
  <audio controls>
      <source src="synthesize-text-audio.mp3" />
  </audio>
  </body>
</html>
```

the <audio> tag is important in order to play the MP3

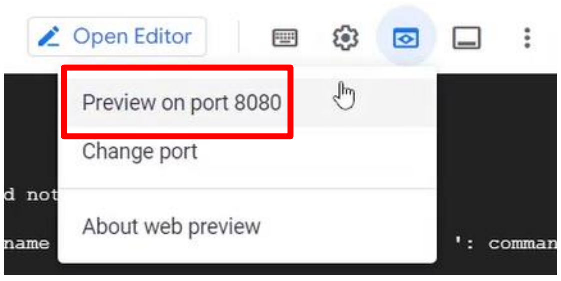
Now go back to the cloud shell and start a simple Python HTTP server with the command shown below:

```
python -m http.server 8080
```

And then click **Web Preview** on the top-right corner of the Cloud Shell



From the menu, select the **Preview on port 8080**



This will open a web page with this structure:

Cloud Text-to-Speech codelab

Output from synthesizing text:

You can now play the audio and hear what you had previously written in the file *synthesize-text.json.*

Task #6 : Create synthetic speech from SSML

As mentioned at the beginning, in addition to a text file, it is possible to use a Speech Synthesis Markup Language (SSML) as input to the API. This allows other properties of the audio to be more precisely defined and controlled.
As before, you have to create a request to the Text-to-Speech API via a json file, let's call it *Synthesize-ssml.json.* As always:

```
touch synthesize-ssml.json
```

And then edit it

```
nano synthesize-ssml.json
```

by entering the following code

```
{
      'input':{
      'ssml':'<speak><s>
            <emphasis level="moderate">Cloud Text-to-Speech
API</emphasis>
            allows developers to include natural-sounding
            <break strength="x-weak"/>
            synthetic human speech as playable audio in their
            applications.</s>
            <s>The Text-to-Speech API converts text or
            <prosody rate="slow">Speech Synthesis Markup
Language</prosody>
            <say-as interpret-as=\"characters\">SSML</say-as>
            input into audio data
            like <say-as interpret-as=\"characters\">MP3</say-as> or
            <sub alias="linear sixteen">LINEAR16</sub>
            <break strength="weak"/>
            (the encoding used in
            <sub alias="wave">WAV</sub> files).</s></speak>'
      },
      'voice':{
      'languageCode':'en-gb',
      'name':'en-GB-Standard-A',
      'ssmlGender':'FEMALE'
      },
      'audioConfig':{
      'audioEncoding':'MP3'
      }
}
```

Note that this contains more information than the plain text file used previously. In particular the *input* section/object contains an *ssml* field.
As discussed at the beginning the *ssml* field contains XML-formatted content the *<speak>* element as its root. Each of the elements present in this XML representation of the input affects the output of the synthetic speech, specifically:

- <s> contains a sentence
- <emphasis> adds stress on the enclosed word or phrase.
- <break> inserts a pause in the speech.
- prosody> customizes the pitch, speaking rate, or volume of the enclosed text, as specified by the rate, pitch, or volume attributes.
- <say-as> provides more guidance about how to interpret and then say the enclosed text, for example, whether to speak a sequence of numbers as ordinal or cardinal.
- <sub> specifies a substitution value to speak for the enclosed text.

Now, in the Cloud Shell call the Text-to-Speech API as has been done before.

```
curl -H "Authorization: Bearer "$(gcloud auth application-default print-access-token) \
  -H "Content-Type: application/json; charset=utf-8" \
  -d @synthesize-ssml.json
"https://texttospeech.googleapis.com/v1/text:synthesize" \
  > synthesize-ssml.txt
```

Now you have to decode the output in order to hear it, use this command to create the corresponding mp3 file *synthesize-ssml-audio.mp3* using the *tts_decode.py* which you have already defined

```
python tts_decode.py --input "synthesize-ssml.txt" --output "synthesize-ssml-audio.mp3"
```

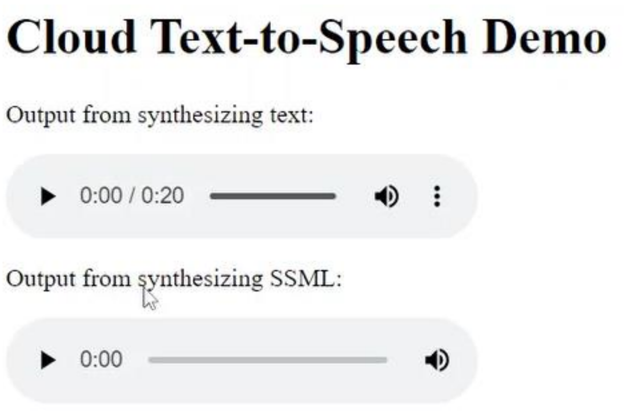
Now add the mp3 file to the *index.html* file you created and used earlier by modifying it as follows:

```
<html>
  <body>
  <h1>Cloud Text-to-Speech Demo</h1>
  <p>
  Output from synthesizing text:
  </p>
  <audio controls>
      <source src="synthesize-text-audio.mp3" />
  </audio>
  <p>
  Output from synthesizing SSML:
  </p>
  <audio controls>
      <source src="synthesize-ssml-audio.mp3" />
  </audio>
  </body>
</html>
```

Now start the Python HTTP server from the Cloud Shell:

```
python -m http.server 8080
```

Click **Web Preview** on the top-right corner of the Cloud Shell and then,rom the menu, select the **Preview on port 8080.**
You should see something like this:


Cloud Text-to-Speech Demo

Output from synthesizing text:

Output from synthesizing SSML:

The first audio is the one created earlier concerning the text.
The second is the one just created via ssml.
If you notice, the two audios are different; specifically the second one adds pauses and different pronunciations.

Task #7 : Configure audio output and device profiles

Now you can provide even more customization to your synthetic speech output created by the Text-to-Speech API.
As always, build your request by creating (touch) and editing (nano) a file that this time we will call *synthesize-with-settings.json.*
When editing put this settings in the file:

```
{
      'input':{
      'text':'The Text-to-Speech API is ideal for any application
            that plays audio of human speech to users. It allows you
            to convert arbitrary strings, words, and sentences into
            the sound of a person speaking the same things.'
      },
      'voice':{
      'languageCode':'en-us',
      'name':'en-GB-Standard-A',
      'ssmlGender':'FEMALE'
      },
      'audioConfig':{
      'speakingRate': 1.15,
      'pitch': -2,
      'audioEncoding':'OGG_OPUS',
      'effectsProfileId': ['headphone-class-device']
      }
}
```

Let us analyse what we have added compared to before:

- The *speakingRate* field specifies a speed at which the speaker says the voice. A value of 1.0 is the normal speed for the voice, 0.5 is half that fast, and 2.0 is twice as fast.
- The *pitch* field specifies a difference in tone to speak the words. The value here specifies a number of semitones lower (negative) or higher (positive) to speak the words.
- The *audioEncoding* field specifies the audio encoding to use for the data. The accepted values for this field are LINEAR16, MP3, and OGG_OPUS.
- The *effectsProfileId* field requests that the Text-to-Speech API optimise the audio output for a specific playback device. The API applies an predefined audio profile to the output that enhances the audio quality on the specified class of devices.

Now follow the same procedure as in the previous two cases
Call the Text-to-Speech API using the curl command:

```
curl -H "Authorization: Bearer "$(gcloud auth application-default print-
access-token) \
  -H "Content-Type: application/json; charset=utf-8" \
  -d @synthesize-with-settings.json
"https://texttospeech.googleapis.com/v1beta1/text:synthesize" \
  > synthesize-with-settings.txt
```

The output of this call is saved to a file called *synthesize-with-settings.txt*.

Now generate the audio file named s*ynthesize-with-settings-audio.mp3*

```
python tts_decode.py --input "synthesize-with-settings.txt" --output
"synthesize-with-settings-audio.ogg"
```

You have to edit the index.html file in order to show and listen the audio that the API have produced

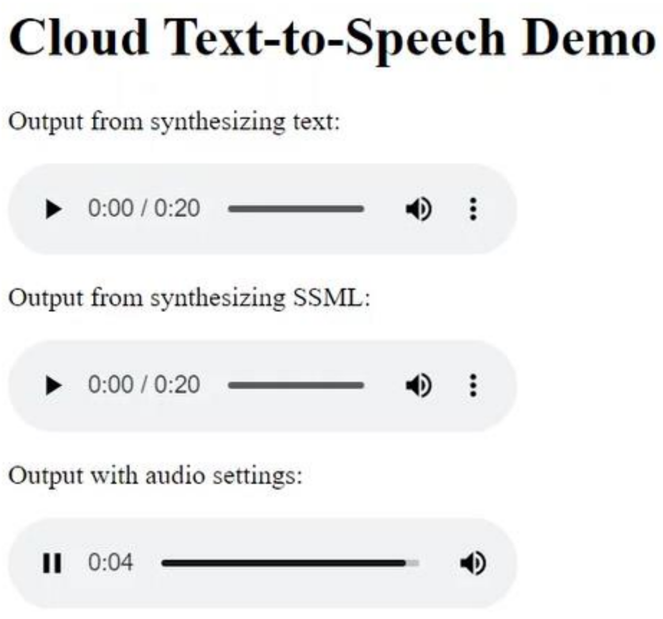
```
<html>
  <body>
  <h1>Cloud Text-to-Speech Demo</h1>
  <p>
  Output from synthesizing text:
  </p>
  <audio controls>
      <source src="synthesize-text-audio.mp3" />
  </audio>
  <p>
  Output from synthesizing SSML:
  </p>
  <audio controls>
      <source src="synthesize-ssml-audio.mp3" />
  </audio>
  </body>
  <p>
  Output with audio settings:
  </p>
  <audio controls>
      <source src="synthesize-with-settings-audio.ogg" />
  </audio>
</html>
```

Restart the Python HTTP server from the Cloud Shell

```
python -m http.server 8080
```

Click **Web Preview** on the top-right corner of the Cloud Shell and then,rom the menu, select the **Preview on port 8080.**

You should see something like this:


Cloud Text-to-Speech Demo

Output from synthesizing text:

Output from synthesizing SSML:

Output with audio settings:

Play the third embedded audio file. Notice that the voice on the audio speaks a bit faster and lower than the previous examples.

Task #8 : Change language

In addition to the English we have used so far, you can use all the different languages and voices visible via the command

```
curl -H "Authorization: Bearer "$(gcloud auth application-default print-access-token) \
      -H "Content-Type: application/json; charset=utf-8" \
      "https://texttospeech.googleapis.com/v1/voices"
```

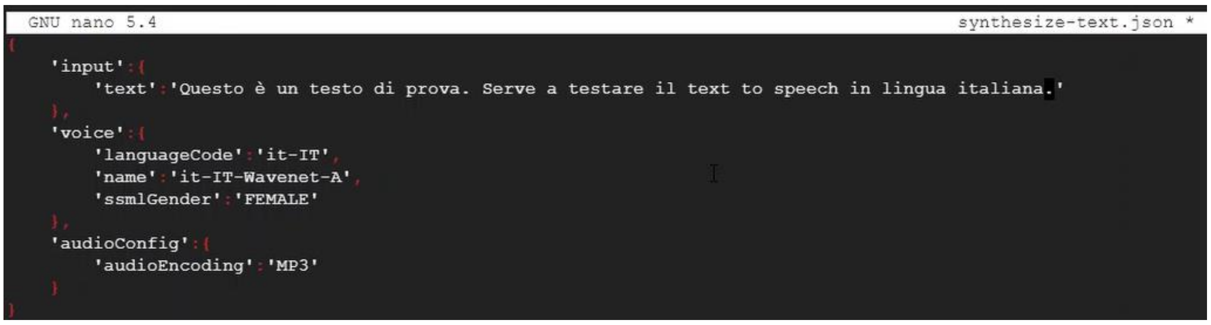
Now we will try the Italian language.

With the following command you can filter voices according to language

```
curl -H "Authorization: Bearer "$(gcloud auth application-default print-access-token) \
      -H "Content-Type: application/json; charset=utf-8" \
      "https://texttospeech.googleapis.com/v1/voices?language_code=it"
```

Choose a voice and replace its properties (languageCodes, name, ssmlGender, naturalSampleRateHertz) in one of the json files that you have created before (for example in the first one, *synthesize-text.json*). Of course, you can also create a new file using the same procedure as in the three previous cases.

The following is an example of editing the very first json file.



```
GNU nano 5.4                                                          synthesize-text.json *
{
    'input':{
        'text':'Questo è un testo di prova. Serve a testare il text to speech in lingua italiana.'
    },
    'voice':{
        'languageCode':'it-IT',
        'name':'it-IT-Wavenet-A',
        'ssmlGender':'FEMALE'
    },
    'audioConfig':{
        'audioEncoding':'MP3'
    }
}
```

Note that we have changed the above-mentioned properties to set the Italian language

Now you have to call the Text-to-Speech API, so run this command:

```
curl -H "Authorization: Bearer "$(gcloud auth application-default print-
access-token) \
  -H "Content-Type: application/json; charset=utf-8" \
  -d @synthesize-text.json
"https://texttospeech.googleapis.com/v1/text:synthesize" \
  > synthesize-text.txt
```

Note: If you named the json file differently, put that name instead of *synthesize-text.json* and put [*your-new-file-name.txt]* in the last row instead of *synthesize-text.txt*

Now, to create an audio file from the response you received from the Text-to-Speech API, run the following command from Cloud Shell, using the *tts_decode.py* file:

```
python tts_decode.py --input "synthesize-text.txt" --output "synthesize-
text-audio.mp3"
```

Note: As before, if you have created a new txt file, use its name instead of *synthesize-text.txt* shown in the first row, and if you want to name the MP3 file differently, put *[your-new-file-name.mp3]* in the last row instead of *synthesize-text-audio.mp3.*

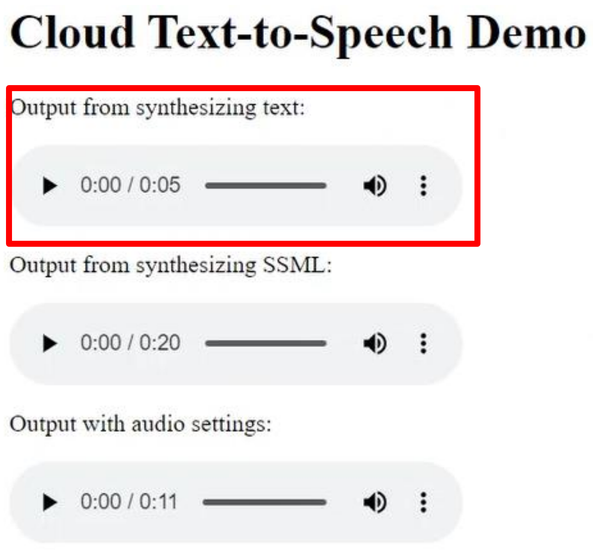
Note: Edit the *index.html* file changing the *source* tag changing the file name if you named the MP3 file differently than *synthesize-text-audio.mp3*.
If you have kept the name *synthesize-text.txt* you will have modified the very first audio file you have created and which will be displayed in the web page at the top of the list of the three audios.

Restart the Python HTTP server from the Cloud Shell

```
python -m http.server 8080
```

Click **Web Preview** on the top-right corner of the Cloud Shell and then, from the menu, select the **Preview on port 8080.**

You should see something like this:


Cloud Text-to-Speech Demo

Output from synthesizing text:

Output from synthesizing SSML:

Output with audio settings:

If you have modified the first file, you will see, as in my case surrounded by red, an audio of a different duration than before. If you play it, you will be able to hear an Italian voice reproduce the text written before.