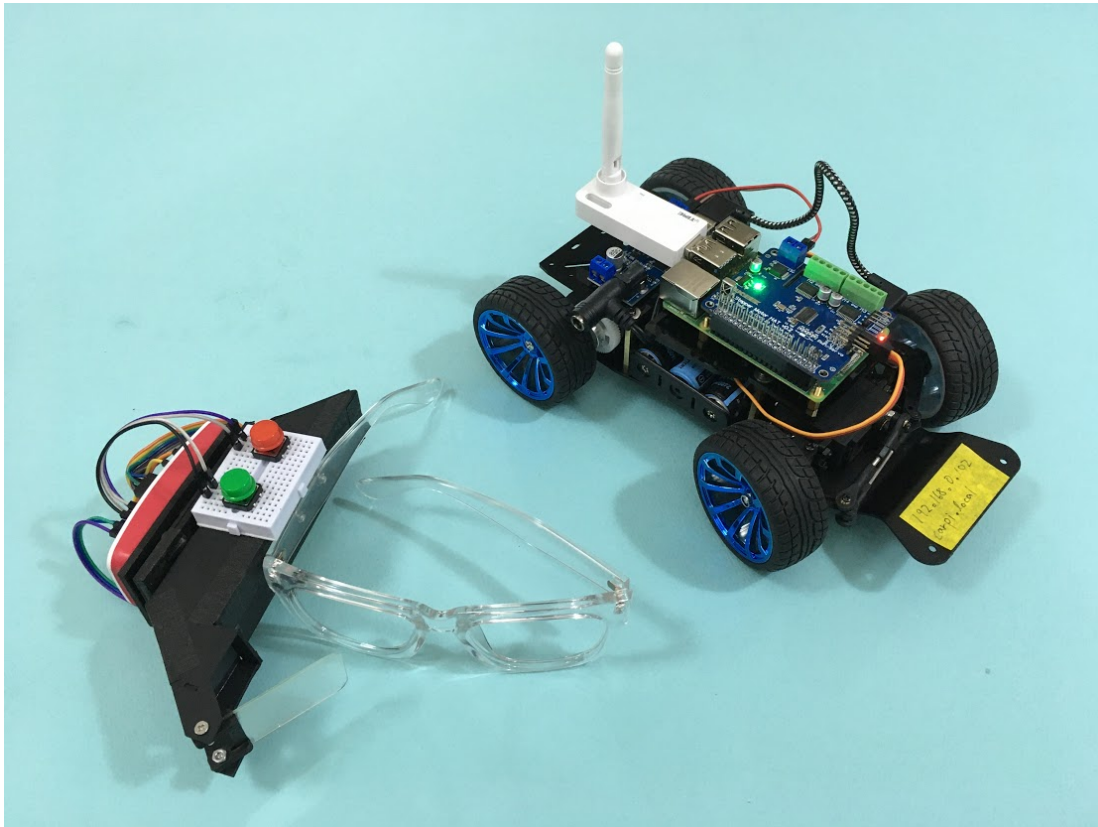


Day3. speech-control-car

예시



기능명세

- AR goggle의 마이크를 통해 음성으로 지령하여 차량을 제어할 수 있도록 한다.
- 첫번째 버튼(모드 버튼)을 누르면 시계 > 날짜 > 차량음성제어 > 다시 시계의 순서로 모드 전환 된다.
- 음성제어모드에서 두번째 버튼(액션 버튼)을 눌러 차량에 명령어 전송을 시작하며, 다시 동일한 버튼을 눌러 전송을 중지할 수 있도록 한다.
- 시계모드에서 두번째 버튼은 아무런 작용하지 않는다.
- 차량제어가 작동하는 동안에는 변환된 텍스트를 oled 디스플레이에 표시하여 사용자에게 작동 상황을 피드백하도록 한다.
- Websocket protocol을 구현한 python websocket modules를 사용하여 goggle과 차량간에 통신한다.
- 앞서 구현한 8개 명령어 (앞으로, 뒤로, 정지, 빠르게, 느리게, 오른쪽, 왼쪽 중앙)는 기본적으로 사용할 수 있도록 한다.
- 콘솔에서 프로그램을 실행시키지 않아도 전원을 켜면 곧바로 작동하여 독립된 장치로 기능할 수 있도록 한다.
- 본인의 아이디어를 더해 발전시켜 완성한다.

Google cloud speech API 를 사용한 음성인식

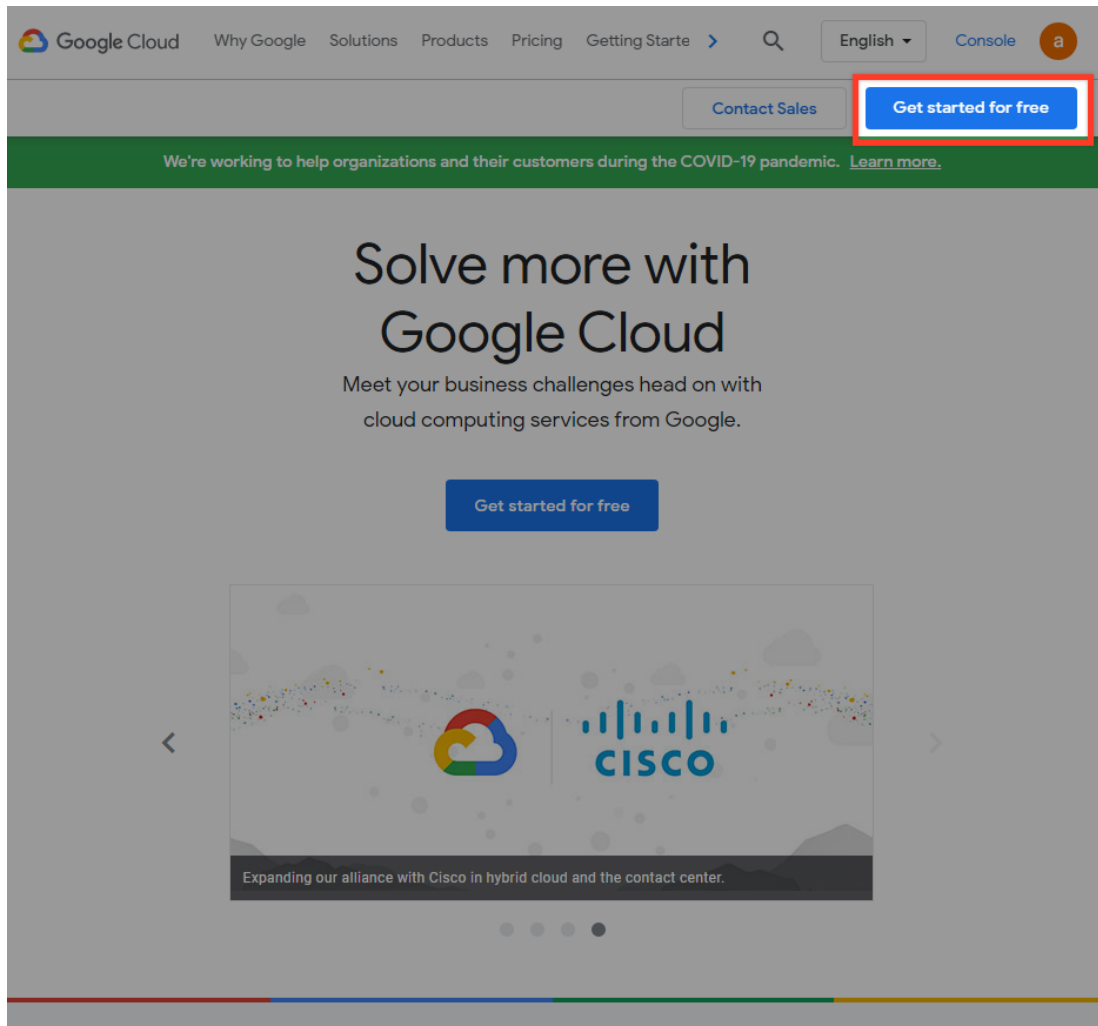
구글 공식 문서: <https://cloud.google.com/speech-to-text/docs> 파이썬 client library 소스: <https://github.com/googleapis/python-speech>

여기서는 먼저 구글에서 제공하는 음성인식 샘플을 작동시켜본다.

GCP (google cloud platform) 프로젝트 생성하기


1. google cloud platform 가입

google cloud speech는 구글 클라우드 플랫폼 서비스의 일부이다. 먼저 구글 계정을 만들어 google cloud platform에 가입하고 프로젝트를 새로 생성한다. <http://www.google.com/cloud> 기존의 gmail 계정을 사용해도 되며, 무료로 가입해 대부분의 기본 기능을 마음껏 사용할 수 있다.



Try Google Cloud Platform for free

Step 1 of 2

 **august park**
08.park@gmail.com

[SWITCH ACCOUNT](#)

Country

South Korea

Terms of Service

☒ I have read and agree to the [Google Cloud Platform Free Trial Terms of Service](#).

Required to continue

CONTINUE

[Privacy policy](#) | [FAQs](#)

Access to all Cloud Platform Products

Get everything you need to build and run your apps, websites and services, including Firebase and the Google Maps API.

\$300 credit for free

Sign up and get \$300 to spend on Google Cloud Platform over the next 12 months.

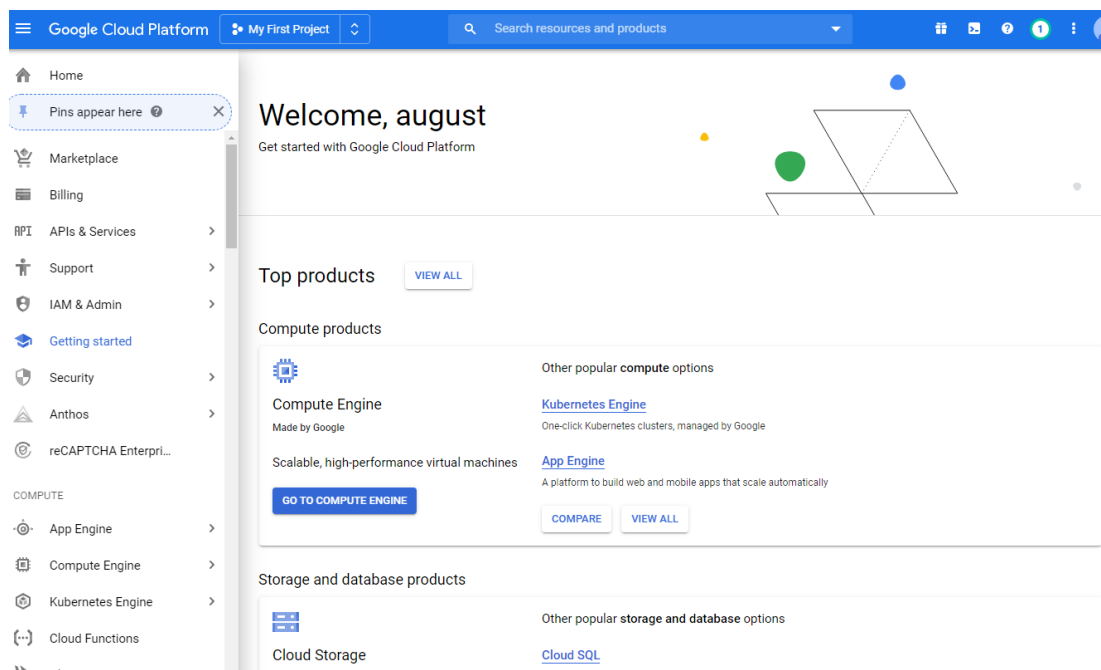
No autocharge after free trial ends

We ask you for your credit card to make sure you are not a robot. You won't be charged unless you manually upgrade to a paid account.

Note: 가끔 Continue 버튼이 눌리지 않는 오류가 보고되었다. 아래 문서를 참고한다. <https://private-space.tistory.com/41>

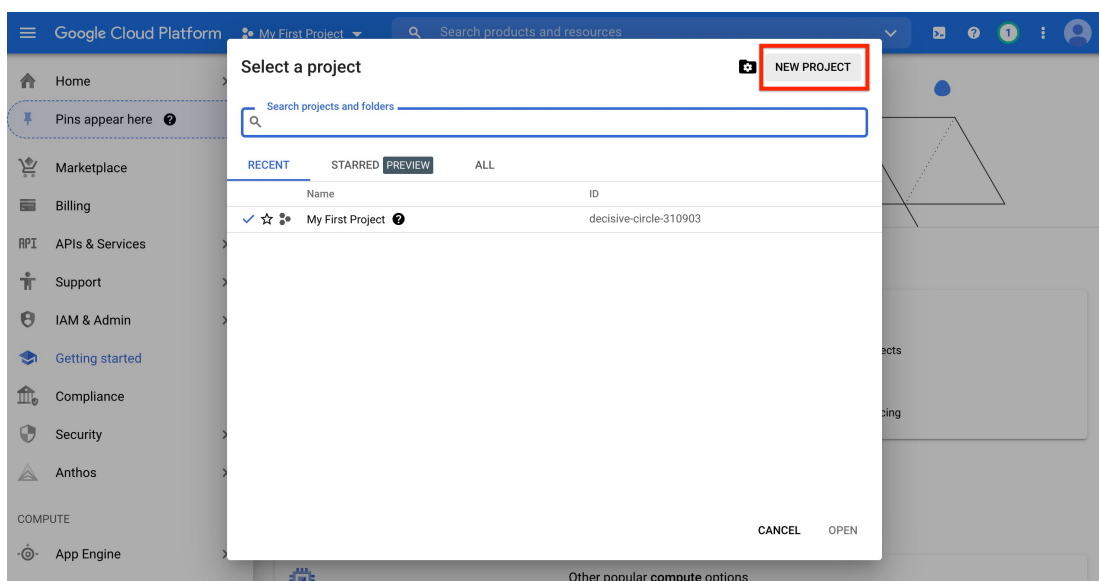
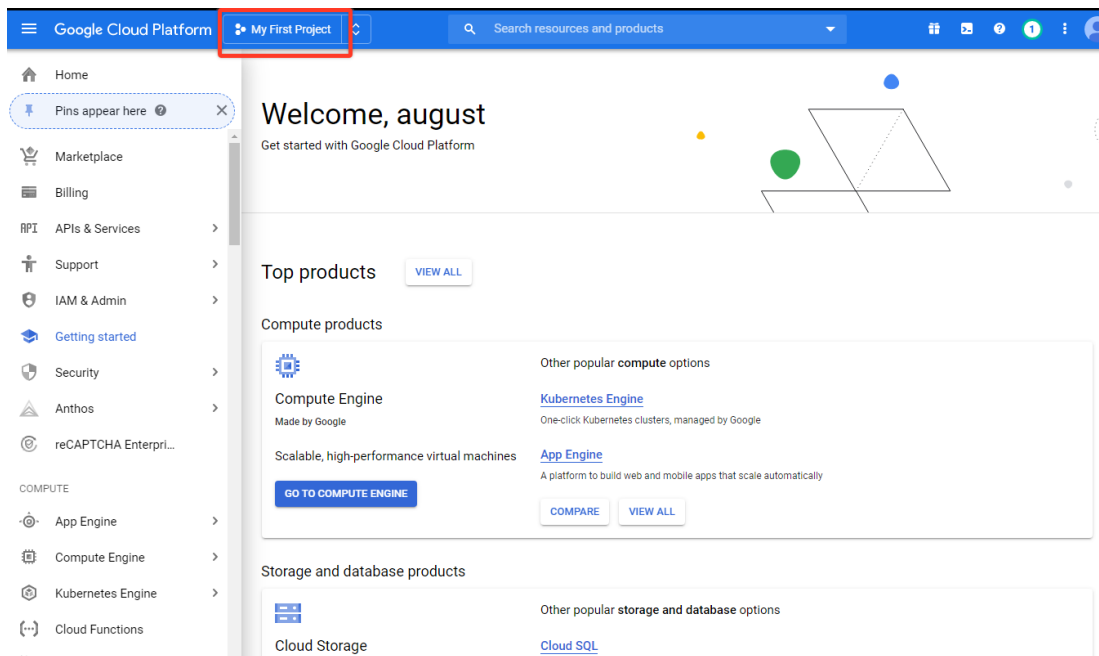
요청하는 정보를 차례로 기입한다. 스마트폰을 통한 사용자 확인 절차가 있으며, 신용카드나 체크카드를 사용해 지불계정을 등록 해 두어야 한다. (결제가 되는 것은 아니므로 안심하자)

완료되면 아래와 같이 cloud platform 초기 화면을 볼 수 있다.



2. 새 프로젝트 생성

구글 클라우드 플랫폼 콘솔화면에서 새로운 프로젝트를 생성한다. <https://console.cloud.google.com/home/dashboard>



프로젝트 명을 입력한다.

Google Cloud Platform

Search products and resources

New Project

You have 11 projects remaining in your quota. Request an increase or delete projects. [Learn more](#)

[MANAGE QUOTAS](#)

Project name *

ssafy-embedded-project

?

Project ID: ssafy-embedded-project-310904. It cannot be changed later. [EDIT](#)

Location *

No organization

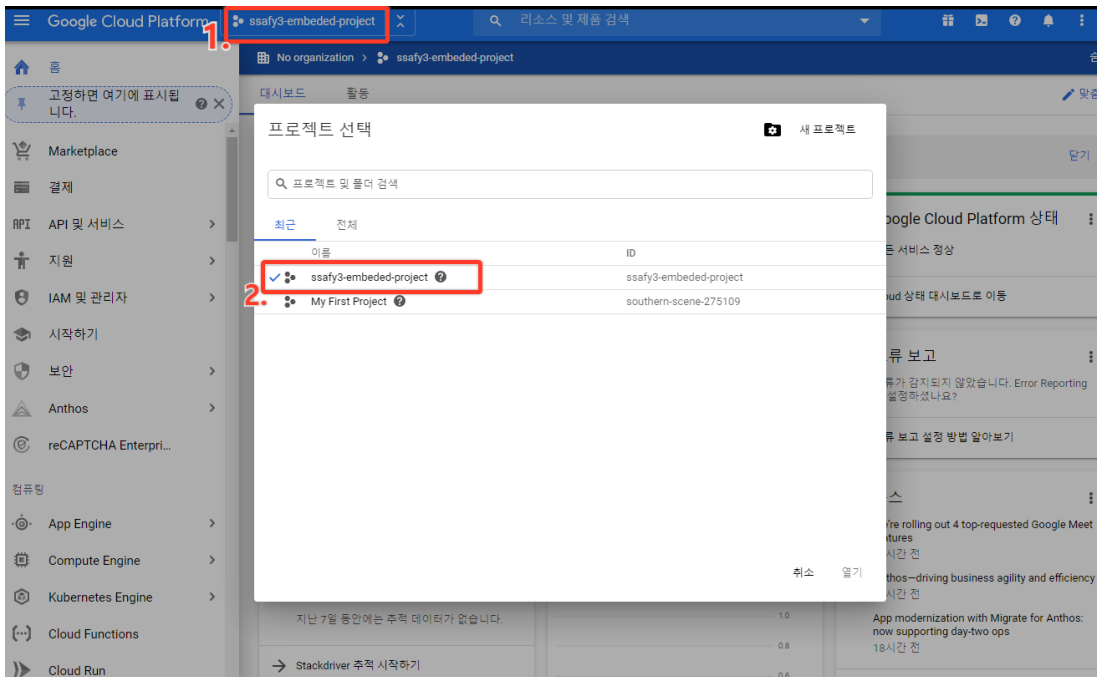
BROWSE

Parent organization or folder

CREATE

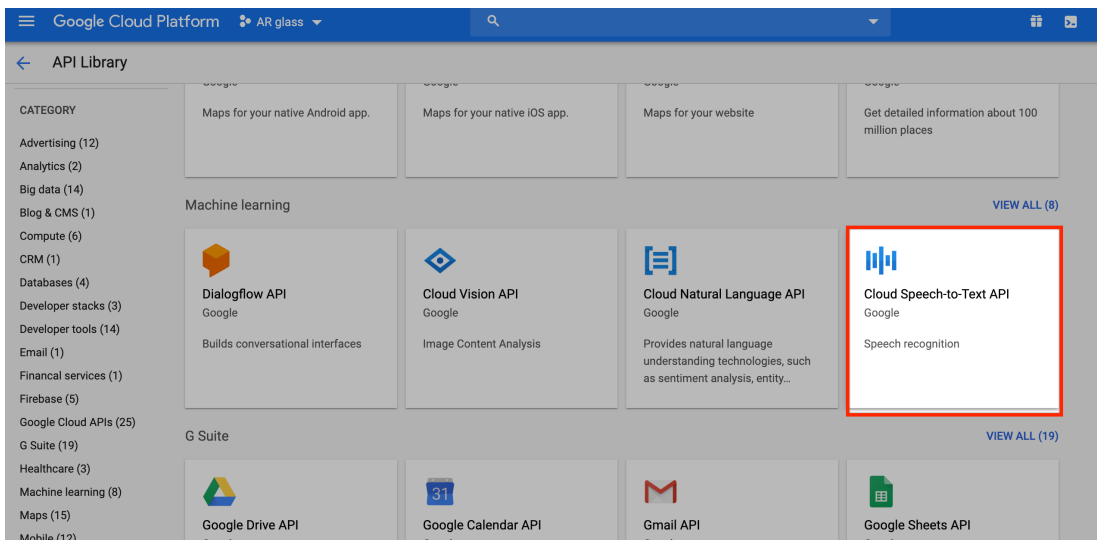
CANCEL

방금 새로 만든 프로젝트를 선택한다.



3. speech-to-text API 켜기

API 및 서비스 > 라이브러리 (<https://console.developers.google.com/apis>)에서 speech-to-text api를 Enabled 시켜준다.



4. Service account key 생성

메뉴 > APIs & Services > 사용자 인증정보(Credentials) > +사용자 인증 정보 만들기(Create credentials) 에서 Service account 추가

Google Cloud Platform ssafy-embedded-project Search products and resources

APIs & Services Cloud Speech-to-Text ...

Overview Metrics Quotas Credentials Data logging

Credentials + CREATE CREDENTIALS DELETE

Credentials console

To view all credentials

Remember this

Help me choose

OAuth 2.0 Client IDs

Service Accounts

Manage service accounts

OAuth client ID

Requests user consent so your app can access the user's data

Service account

Enables server-to-server, app-level authentication using robot accounts

CONFIGURE CONSENT SCREEN

OAuth 2.0 Client IDs

Name	Creation date	Type	Client ID
No OAuth clients to display			

Service Accounts

Email	Name
No service accounts to display	

Google Cloud Platform ssafy-embedded-project Search products and resources

IAM & Admin

Identity & Organization Policy Troubleshooter Policy Analyzer Organization Policies Service Accounts Labels Tags Settings Privacy & Security Identity-Aware Proxy Roles Audit Logs

Create service account

1 Service account details

Service account name

ssafy-arglass

Display name for this service account

Service account email

ssafy-arglass @ssafy-embedded-project-310904.iam.gserviceaccount.com

Service account description

arglass device 사용 계정

Describe what this service account will do

CREATE

2 Grant this service account access to project (optional)

3 Grant users access to this service account (optional)

DONE CANCEL

서비스 계정 이름 짓기

역할(Role)은 Owner로 지정.

Google Cloud Platform ssafy-embedded-project Search products and resources

IAM & Admin

Identity & Organization Policy Troubleshooter Policy Analyzer Organization Policies Service Accounts Labels Tags Settings Privacy & Security Identity-Aware Proxy Roles Audit Logs Essential Contacts Groups Early Access Center Quotas

Create service account

Service account details

2 Grant this service account access to project (optional)

Grant this service account access to ssafy-embedded-project so that it has permission to complete specific actions on the resources in your project. [Learn more](#)

Select a role

Condition

Type to filter

Quick access	Browser
Currently used	Editor
Basic	Owner
All roles	Viewer
Access Approval	
Access Context Manager	
Actions	

MANAGE ROLES

Owner

Full access to all resources.

키파일(.json) 을 만들고 다운받아 /home/pi로 옮긴다.

The screenshot shows the Google Cloud Platform IAM & Admin console. The left sidebar lists various IAM & Admin tools, with 'Service Accounts' selected. The main content area displays 'Service accounts for project "ssafy-embedded-project"'. A table lists the service accounts, with one account highlighted: 'ssafy-arglass@ssafy-embedded-project-310904.iam.gserviceaccount.com'. The account is in a 'Created' state (green checkmark) and has a description 'ar glass 사용 계정'.

Email	Status	Name	Description	Key ID	Key creation date	Actions
ssafy-arglass@ssafy-embedded-project-310904.iam.gserviceaccount.com	Created	ssafy-arglass	ar glass 사용 계정	No keys		

The screenshot shows the Google Cloud Platform IAM & Admin console, specifically the 'Keys' tab for the 'ssafy-arglass' service account. The 'Keys' tab is selected, and the 'Create new key' button is highlighted. The 'Create new key' button is located under the 'ADD KEY' dropdown menu.

ADD KEY ▾

Create new key

Upload existing key

Create private key for "ssafy-arglass"

Downloads a file that contains the private key. Store the file securely because this key can't be recovered if lost.

Key type

☒ JSON

Recommended

☐ P12

For backward compatibility with code using the P12 format

CANCEL

CREATE

```
C:/> scp xxxxxxxx-xxxxxxxxxx.json pi@raspberrypi.local:~/
```

5. GOOGLE_APPLICATION_CREDENTIALS 환경변수 설정

```
export GOOGLE_APPLICATION_CREDENTIALS="/home/pi/xxxxxxx-xxxxxxxxx.json"
```

google cloud speech python 라이브러리

- 라이브러리 설치

```
$ pip3 install google-cloud-speech
```

- 샘플코드 실행 google-cloud-speech python library 공식 레퍼런스:
<https://googleapis.dev/python/speech/latest/index.html>
- 구글 클라우드 스피치는 3가지 방법으로 음성 전달/ 결과 받기 할 수 있는데,
 - 짧은 음성 데이터(1분미만)을 보내고 기다렸다가 결과를 받아서 활용. - SpeechClient.recognize()사용
 - 긴 음성 데이터를 보내놓고 해석이 되는데로 조금씩 결과를 받아오는 방식 -
SpeechClient.long_running_recognize() 사용
 - 마이크를 사용해 실시간으로 데이터를 보내면서 즉각적으로 결과를 받아오는 방식 -
SpeechClient.streaming_recognizer()사용.

여기서는 1번과 3번 방식의 예를 보자 참고: <https://cloud.google.com/speech-to-text/docs/basics>

샘플 코드 (짧은 음성파일)

소스파일: https://github.com/ssafy-embedded-project/google_speech_samples/raw/main/google_speech_from_file.py

```
# google_speech_from_file.py
from google.cloud import speech
import io

# 사용할 파일 위치
local_file_path = '/home/pi/test.wav'

# Instantiates a client
client = speech.SpeechClient()

# 리퀘스트 구성
config = speech.RecognitionConfig(
    encoding=speech.RecognitionConfig.AudioEncoding.LINEAR16,
    sample_rate_hertz=48000,
    language_code="ko-KR", # 영어라면 en-US
)

with io.open(local_file_path, "rb") as f: # 오디오 파일. 바이너리모드로 읽는다.
    content = f.read()
audio = speech.RecognitionAudio(content=content)

# 리퀘스트 보내고 응답 받아온다.
```



```
response = client.recognize(config=config, audio=audio)
```

```
for result in response.results: # results 는 음성이 잠시 멈출 때로 구분되는 여러개의  
세그먼트의 리스트.( 한 문장 ~= 한 세그먼트)  
    print(f"Transcript: {result.alternatives[0].transcript}") # 각 세그먼트의  
alternatives항목은 복수개의 제안일 수 있으며, 가능성이 높은 순으로 정렬되므로 대부분의 경우  
alternatives[0]을 사용한다.
```

음성파일이 포함된 리퀘스트를 보내면 해석이 포함된 응답을 받는다.

리퀘스트 포맷 (json)

```
// 로컬파일이 아닌 구글 클라우드 스토리지상의 파일을 전달하는 경우의 리퀘스트 예  
{  
  "config": {  
    "encoding": "FLAC",  
    "sampleRateHertz": 16000,  
    "languageCode": "en-US",  
  },  
  "audio": {  
    "uri": "gs://cloud-samples-tests/speech/brooklyn.flac"  
  }  
}
```

- config: 인코딩 형식 등의 메타정보
- encoding: 선택된 파일의 인코딩 형식. 'LINEAR16','FLAC'등 무손실 인코딩을 추천함. ('wav'파일은 보통 'LINEAR16' 인코딩이다.) 'MP3', 'OGG-OPUS'등도 사용 가능. 참고: <https://cloud.google.com/speech-to-text/docs/encoding>
- sampleRateHertz: 음원파일의 샘플링 레이트. 음원 녹음시 16000Hz 이상의 sampling rate를 추천함. 낮은rate의 파일을 높은rate로 다시 인코딩하는 것은 소용 없음.
- languageCode: 한국어는 "ko-KR"
- audio: 음성데이터
- url: 음성데이터의 url. 현재로서는 google cloud storage(gs://....)상의 파일만 사용가능하다고 함.
- content: 오디오데이터. 로컬파일을 사용한다면 url 대신 content 매개변수를 사용한다.

응답 포맷 (json)

```
{  
  "name": "6212202767953098955",  
  "metadata": {  
    "@type":  
"type.googleapis.com/google.cloud.speech.v1.LongRunningRecognizeMetadata",  
    "progressPercent": 100,  
    "startTime": "2017-07-24T10:21:22.013650Z",  
    "lastUpdateTime": "2017-07-24T10:21:45.278630Z"  
  },  
}
```

```

"done": true,
"response": {
  "@type":
"type.googleapis.com/google.cloud.speech.v1.LongRunningRecognizeResponse",
  "results": [
    {
      "alternatives": [
        {
          "transcript": "Four score and twenty...(etc)...",
          "confidence": 0.97186122,
          "words": [
            {
              "startTime": "1.300s",
              "endTime": "1.400s",
              "word": "Four"
            },
            {
              "startTime": "1.400s",
              "endTime": "1.600s",
              "word": "score"
            },
            {
              "startTime": "1.600s",
              "endTime": "1.600s",
              "word": "and"
            },
            {
              "startTime": "1.600s",
              "endTime": "1.900s",
              "word": "twenty"
            },
            ...
          ]
        }
      ]
    },
    {
      "alternatives": [
        {
          "transcript": "for score and plenty...(etc)...",
          "confidence": 0.9041967,
        }
      ]
    }
  ]
}

```

- results: 음성 인식된 결과 세그먼트의 리스트.
- alternatives: 주어진 음성이 두가지 이상으로 해석될 수 있기 때문에 ('삼식이'vs'산지기') 복수개의 alternatives를 받아보도록 할 수 있다. 확실성이 높은 순으로 정렬되어오므로 일반적으로 alternatives[0]을 사용하면 된다.
- confidence: 확실성 0~1
- transcript: 해석된 text

마이크 사용 (실시간 음성인식) 예제

마이크에 말하면 실시간으로 text로 바꾸어주는 기본 예제 실행해보자. 소스파일: https://github.com/ssafy-embedded-project/google_speech_samples/raw/main/transcribe_streaming_mic.py

```
# transcribe_streaming_mic.py

import re # 정규표현식 모듈
import sys

from google.cloud import speech
import pyaudio # 파이썬에서 오디오 입력 사용
import queue

# Audio recording parameters
RATE = 16000
CHUNK = int(RATE / 10)

class MicrophoneStream(object):
    def __init__(self, rate, chunk):
        self._rate = rate
        self._chunk = chunk

        # Create a thread-safe buffer of audio data
        self._buff = queue.Queue() # pyaudio가 전달해주는 데이터를 담을 큐
        self.closed = True

        # 파이썬 context manager사용. 여기에서는 실행중 문제가 발생해도 오디오장치를 제대로 닫도
        # 록 할 수 있기 위함.
        def __enter__(self):
            self._audio_interface = pyaudio.PyAudio() # 시작할 때 pyaudio 데이터
            스트림 열림
            self._audio_stream = self._audio_interface.open( # pyaudio.open()은
            pyaudio.Stream object를 리턴.
                format=pyaudio.paInt16, # 16bit 다이내믹 레인지
                channels=1,
                rate=self._rate,
                input=True,
                frames_per_buffer=self._chunk,
                stream_callback=self._fill_buffer, # pyaudio에서 한 블록의 데이터가
                들어올 때 호출되는 콜백
            )

            self.closed = False

            return self

        def __exit__(self, type, value, traceback):
            self._audio_stream.stop_stream()
            self._audio_stream.close()
            self.closed = True
            self._buff.put(None)
            self._audio_interface.terminate() # 끝날 때 반드시 pyaudio 스트림 닫도
            록 한다.

        def _fill_buffer(self, in_data, frame_count, time_info, status_flags):
            # pyaudio.Stream에서 호출되는 콜백은 4개 매개변수 갖고, 2개값 리턴한다. pyaudio문서 참고.
```

```

self._buff.put(in_data) # 큐에 데이터 추가
return None, pyaudio.paContinue

# 한 라운드의 루프마다 현재 버퍼의 내용을 모아서 byte-stream을 yield함.
def generator(self):
    while not self.closed:
        # Use a blocking get() to ensure there's at least one chunk of
        # data, and stop iteration if the chunk is None, indicating
        # the
        # end of the audio stream.
        chunk = self._buff.get()
        if chunk is None:
            return
        data = [chunk]

        # Now consume whatever other data's still buffered.
        while True:
            try:
                chunk = self._buff.get(block=False) # 가장 오래된 데이터부
                # the
                # 순차적으로 data[]에 추가함.
                if chunk is None:
                    return
                data.append(chunk)
            except queue.Empty: # 큐에 더이상 데이터가 없을 때까지
                break

        yield b''.join(data) # byte-stream

# response 화면에 출력
def listen_print_loop(responses):
    num_chars_printed = 0
    for response in responses:
        if not response.results:
            continue

        # The `results` list is consecutive. For streaming, we only care
        # about
        # the first result being considered, since once it's `is_final`,
        # it
        # moves on to considering the next utterance.
        # 최종적인 결과값은 언제나 results[0]에 반영되므로 result[0]만 고려.
        result = response.results[0]
        if not result.alternatives:
            continue

        # 확실성 가장 높은 alternative의 해석
        transcript = result.alternatives[0].transcript

        # 완성된 문장이 intrim 문장보다 짧다면, 나머지 부분은 ' '으로 overwrite해 가려준
        # 다.
        overwrite_chars = ' ' * (num_chars_printed - len(transcript))

        if not result.is_final: # 확정된 transcript가 아니라면,
            sys.stdout.write(transcript + overwrite_chars + '\r') #
            # '\r'로 줄바꿈은 하지 않고 맨 앞으로 돌아가 이전 문장위에 덧쓰도록 한다.
            sys.stdout.flush()

```

```

num_chars_printed = len(transcript)

else: # 확정된 transcript라면
    print(transcript + overwrite_chars)

    # 문장중에 '명령끝'이라는 단어가 있다면 종료한다.
    if re.search(r'\b(명령 끝)\b', transcript, re.I):
        print('Exiting..')
        break

num_chars_printed = 0

def main():
    # 한국말 사용
    language_code = 'ko-KR' # a BCP-47 language tag

    client = speech.SpeechClient()
    config = speech.RecognitionConfig(
        encoding='LINEAR16', #
enums.RecognitionConfig.AudioEncoding.LINEAR16
        sample_rate_hertz=RATE,
        max_alternatives=1, # 가장 가능성 높은 1개 alternative만 받음.
        language_code=language_code)
    streaming_config = speech.StreamingRecognitionConfig(
        config=config,
        interim_results=True) # 해석완료되지 않은(is_final=false) 중도값도 사용.

    with MicrophoneStream(RATE, CHUNK) as stream: # 사운드 스트림 오브젝트 생
성.
        # pyaudio가 terminate()
        # context manager 사용.
        audio_generator = stream.generator()
        requests =
(speech.StreamingRecognizeRequest(audio_content=content)
        for content in audio_generator) # generator
expression. 요청 생성

    responses = client.streaming_recognize(streaming_config, requests)
# 요청 전달 & 응답 가져옴
    listen_print_loop(responses) # 결과 출력. requests, responses 모두
iterable object

if __name__ == '__main__':
    main()

```

tips

일상적 대화인지, 전화 통화인지, 단문의 명령어인지등에 따라 다른 모델을 사용해 인식율을 높일 수 있다. 참고:

<https://cloud.google.com/speech-to-text/docs/transcription-model>

상황에 따라 자주 사용되는 단어나 문구는 우선순위를 높여 더욱 빠르고 정확하게 인식되도록 할 수 있다. 참고:

<https://cloud.google.com/speech-to-text/docs/speech-adaptation>

기타 다양한 기능소개와 사용법은 문서를 참고한다.: <https://cloud.google.com/speech-to-text/docs>

python threading 모듈

참고: <https://docs.python.org/ko/3.7/library/threading.html>

새로운 스레드를 만들어 기존 스레드와 별도로 동작하게 한다. 주의 할 점은 새로운 스레드를 만든다고해서 실행속도가 2배 빨라지는 것은 아니라는 점. 스레드는 2개라도 파이썬은 언제나 한 순간에 하나의 스레드만 실행하므로 속도는 (오버헤드 감안) 오히려 느려질 수 있다. 속도보다는, 동시에 진행되어야할 2개의 독립적인 태스크를 처리하기 위해 (예를 들어 음성데이터 처리와 동시에 화면 업데이트) 사용한다.

1. 함수로 스레드 만들기 샘플 코드

```
# threading-sample.py
# 1. 모듈 импорт
import threading
import time

# 2. 서브스레드 함수 정의
def sub_thread():
    for var in range(30):
        print(f'we are in sub-thread:{var}')
        var += 1
        time.sleep(1)

# 3. threading.Thread()로 스레드 생성
t = threading.Thread(target = sub_thread)

# 4. 스레드 시작
t.start()

# main thread
text = input('아무키나 누를 때까지 메인스레드는 block됨:')

print('메인 스레드가 계속진행됩니다')
while True:
    print('we are in main...')
    time.sleep(1)
```

asyncio 모듈

threading과는 달리 하나의 스레드에서 여러 작업을 동시적으로 처리 할 수 있는 'coroutine'을 활용한 asyncio는 조금 더 가볍고 스레드 전환에 따르는 오버헤드 부담이 적으며, 여러 스레드에서 하나의 자원에 접근하는 경우의 문제에 신경쓰지 않아도 된다. 특히 네트워크 통신이나 파일입출력등 io작업에 적합하다.

It still takes one execution step at a time. The difference is that the system may not wait for an execution step to be completed before moving on to the next one. This means that the program will move on to future execution steps even though a previous step hasn't yet finished and is still running elsewhere. This also means that the program knows what to do when a previous step does finish running.

참고: <https://realpython.com/python-async-features/> 파이썬 공식문서 참고:
<https://docs.python.org/ko/3.7/library/asyncio-task.html#coroutine>

asyncio 예제

blocking 방식 코드라면

```
# 기존의 blocking 코드
# 2개의 io 묘사

import time

def request1():
    time.sleep(10)
    return (time.perf_counter()) # perf_counter() 퍼포먼스카운터: 프로그램 실행시간

def request2():
    time.sleep(5)
    return(time.perf_counter())

def main():
    print('start...')
    start_time = time.perf_counter()

    resp1_time = request1()
    resp2_time = request2()
    print(f'resp1 takes {resp1_time - start_time}')
    print(f'resp2 takes {resp2_time - start_time}')

main()
```

결과

```
start...
resp1 takes 10.00129740700001
resp2 takes 15.004810797000005
```

acyncio를 사용한 non-blocking 방식 코드

```
# non-blocking이 지원되는 asyncio.sleep을 사용한다면...

import asyncio
import time
```

```

async def request1():
    await asyncio.sleep(10)
    return(time.perf_counter())

async def request2():
    await asyncio.sleep(5)
    return(time.perf_counter())

async def main():
    print('start...')
    start_time = time.perf_counter()

    resp_time = await asyncio.gather(request1(), request2())

    print(f'resp1 takes {resp_time[0] - start_time}')
    print(f'resp2 takes {resp_time[1] - start_time}')

asyncio.run(main())

```

결과

```

start...
resp1 takes 10.004390306000005
resp2 takes 5.004985657000134

```

이벤트 루프 제어

이벤트루프의 종료 조건을 여러가지 정해 줄 수 있다. 참고: <https://docs.python.org/ko/3.7/library/asyncio-task.html#running-tasks-concurrently>

별도의 스레드에서 blocking code 사용하기

Task나 Future 오브젝트를 반환하지 않도록 만들어진, 그래서 asyncio 와 함께 사용할 수 없는 기존의 대부분의 코드들을 asyncio와 함께 non-blocking으로 작동하도록 하고자한다면 별도의 스레드에서 실행토록 한다. 참고:

<https://docs.python.org/ko/3.7/library/asyncio-eventloop.html#executing-code-in-thread-or-process-pools>

websockets

websocket기술 개요

기존 request-respond 방식의 http 프로토콜의 아쉬운점을 보완하기 위해 인터넷환경, 특히 웹브라우저와 서버간에서도 tcp 소켓과 같은 양방향의 지속적인 연결을 구현하기위한 통신프로토콜로 제안되었다.

http와 비교하면 클라이언트의 리퀘스트가 없어도 서버측에서 정보를 보낼 수 있으므로 실시간성이 중요한 경우에도 활용 가능. tcp socket과 비교하면 대부분의 웹브라우저가 기본으로 지원하며 특히 인터넷환경에서 범용적으로 활용가능.

1. websockets 모듈 인스톨

서버측/ 클라이언트측 모두 설치


```
$ pip3 install websockets
```

2. websockets 기본 예제

클라이언트에서 '이름'을 보내면 서버에서 'Hello 이름'을 리턴한다. websockets 공식 레퍼런스 참고:

<https://websockets.readthedocs.io/en/stable/intro.html>

비동기 프로그래밍을 위해 asyncio module을 필요로한다.

```
# websockets basic-example server.py
import asyncio # non-blocking 방식으로 네트워크 통신하기 위해 asyncio 모듈 필요.
import websockets # websockets 모듈 임포트

async def hello(websocket, path): # 아래의 websockets.serve()에 의해 호출되는 웹
    # 소켓 핸들러 코루틴은 websocketServerProtocol 오브젝트와 unix의 소켓위치(파일)을 전달받기로
    # 약속되어있다.
    while True:
        name = await websocket.recv()
        print(f"{name}을 받았습니다.")

        greeting = f"Hello {name}!"
        await websocket.send(greeting)
        print(f"{greeting}을 보냈습니다.")

async def main():
    server = await websockets.serve(hello, host = '192.168.0.112',
    port=4321) # 서버를 만들고 클라이언트가 연결을 요청할 때마다 웹소켓 연결을 만들 어
    websocketServerProtocol object를 hello에 넘겨준다. 리턴값은 websocketServer
    object이다.
    await asyncio.gather(server.wait_closed()) # 명시적으로 close()하지 않는 한 서
    # 버가 계속 작동하도록 한다. wait_closed()가 없다면 클라이언트가 연결해오기도 전에 서버가 종료되
    # 어버린다

    asyncio.run(main())
```

```
# websockets_basic-example_client.py

import asyncio
import websockets

async def hello():
    uri = "ws://192.168.0.105:5678"
    async with websockets.connect(uri) as websocket:
        while True:
            name = input("what's your name? ")

            await websocket.send(name)
            print(f"{name} sent")

            greeting = await websocket.recv()
            print(f"{greeting} received.")
```

```
async def main():
    await asyncio.gather(hello())

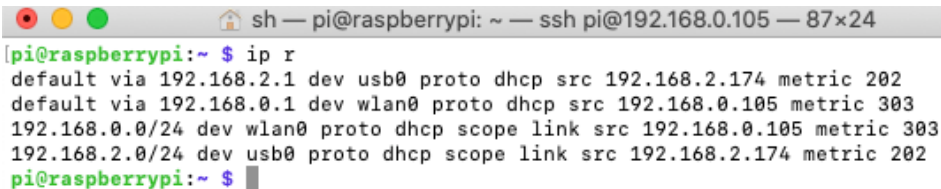
asyncio.run(main())
```

Raspberry pi 고정 ip 설정

라우터 주소와 사용할 ip address 알기

ip 충돌을 방지하기 위해 (유동 ip방식으로 할당된) 현재의 ip 주소를 사용하는 것을 추천한다. 'ip r' 명령어를 통해 현재 ip와 라우터 주소 알아낸다.

```
$ ip r
```



```
sh — pi@raspberrypi: ~ — ssh pi@192.168.0.105 — 87x24
pi@raspberrypi:~ $ ip r
default via 192.168.2.1 dev usb0 proto dhcp src 192.168.2.174 metric 202
default via 192.168.0.1 dev wlan0 proto dhcp src 192.168.0.105 metric 303
192.168.0.0/24 dev wlan0 proto dhcp scope link src 192.168.0.105 metric 303
192.168.2.0/24 dev usb0 proto dhcp scope link src 192.168.2.174 metric 202
pi@raspberrypi:~ $
```

default via 192.168.0.1 dev wlan0 proto dhcp scr 192.168.0.105 metric 303 -> wifi연결에서 라우터 주소 (192.168.0.1)와 현재의 ip(192.168.0.105)를 알 수 있다.

DNS 서버 주소 알기

/etc/resolv.conf 파일에서 확인한다.

```
$ cat /etc/resolv.conf
```

dhcpcd.conf 설정파일 수정

```
$ sudo nano /etc/dhcpcd.conf
```

조금 아래쪽에 보면 주석처리 되어있는 static ip 설정 example이 있다. 여기에 사용할 각각의 주소를 집어넣어준다.

```
sh — pi@raspberrypi: ~ — ssh pi@192.168.0.105 — 87x24
GNU nano 3.2 /etc/dhcpd.conf Modified

# Most distributions have NTP support.
#option ntp_servers

# A ServerID is required by RFC2131.
require dhcp_server_identifier

# Generate SLAAC address using the Hardware Address of the interface
#slaac hwaddr
# OR generate Stable Private IPv6 Addresses based from the DUID
slaac private

# Example static IP configuration:
interface wlan
static ip_address=192.168.0.105/24
#static ip6_address=fd51:42f8:caae:d92e::ff/64
static routers=192.168.0.1
static domain_name_servers=192.168.0.1 8.8.8.8 fd51:42f8:caae:d92e::1

^G Get Help  ^O Write Out  ^W Where Is   ^K Cut Text   ^J Justify    ^C Cur Pos
^X Exit      ^R Read File  ^\ Replace    ^U Uncut Text ^T To Spell   ^_ Go To Line
```

```
interface NETWORK static ip_address=STATICIP/24 static routers=ROUTERIP static
domain_name_servers=DNSIP
```

- NETWORK : eth0 / wlan0 / usb0... 무선연결 사용한다면 wlan0. or wlan1
- STATICIP : 사용할 고정IP 주소
- ROUTERIP : 라우터 주소
- DNSIP : 네임서버주소. 구글(8.8.8.8) 이나 Cloudflare(1.1.1.1) 사용해도 좋다.

리부팅 후 테스트

```
$ sudo reboot
...
```

Raspberry pi 프로그램 자동시작하기

다양한 방법중, 'pi'계정 로그인과 동시에 자동실행되는 `.bashrc`파일을 활용하는 예를 살펴본다.

'~/.bashrc' 파일 사용

부팅과정에서 사용자 로그인 직후에 실행된다. 즉, pi계정으로 실행된다는 점 유의. 파일 내에서 되도록 절대경로를 사용해 혼란을 방지하도록 한다.

```
$ sudo nano ~/.bashrc
...
```

- 뒤에 `&`을 붙여 백그라운드로 실행시키도록 한다. 만일 코드에 문제가 있는데 콘솔 로그인이 안되는 경우, 이러지도 저러지도 못하는 사태가 발생할 수 있다.
- 로그를 기록해서 무슨이 벌어지는지 확인하자 (`2>&1` 참고: <https://www.briantorti.com/understanding-shell-script-idiom-redirect/>)

```
# ~/.bashrc: executed by bash(1) for non-login shells.
# see /usr/share/doc/bash/examples/startup-files (in the package bash-doc)
# for examples
...
# enable programmable completion features (you don't need to enable
# this, if it's already enabled in /etc/bash.bashrc and /etc/profile
# sources /etc/bash.bashrc).
if ! shopt -oq posix; then
  if [ -f /usr/share/bash-completion/bash_completion ]; then
    . /usr/share/bash-completion/bash_completion
  elif [ -f /etc/bash_completion ]; then
    . /etc/bash_completion
  fi
fi

...파일 가장 아래에 추가
# 파이썬 코드 부팅과 함께 자동실행
python3 /home/pi/mycode.py& > /home/pi/log.txt 2>&1
```

Note: google cloud speech API 사용에 필요한 GOOGLE_APPLICATION_CREDENTIALS 환경변수를 파이썬 코드보다 먼저 실행되는 위치에 선언해준다.

```
$ sudo nano /etc/profile

...마지막에 추가
# google GCP service key
export GOOGLE_APPLICATION_CREDENTIALS="/home/pi/google_servicekey.json"
```

추가과제 (optional)

- 프로젝트를 발전시켜 자신만의 기능을 더한다.
- '5m 앞에서 우회전' 등 복합적인 명령을 구현해본다.
- 카메라 혹은 센서를 활용하는 제3의 모드를 구현해본다.