*Kevin Chen*

*Kevinjc.tw@gmail.com*

*部署Python AI 模型在Google App Engine*

*[Document Subtitle]*

|  |  |  |  |
| --- | --- | --- | --- |
| 版本 | 日期 | 說明 | 備註 |
| 1.0 | 2021-07-20 | 初稿 |  |
| 1.1 | 2021-07-26 | 新增app.yaml說明 |  |

Table of Contents

[1. 前置條件 2](#_Toc78196242)

[2.開始設定 2](#_Toc78196243)

[2.1 在cloud shell輸入 gcloud config list project會顯示登入使用者可使用的project list 2](#_Toc78196244)

[2.2 到Github下載您的python 模型訓練 repo ，在cloud shell CD使用者目錄後，輸入 git clone https://github.com/YuehMintTai/RPython.git 3](#_Toc78196245)

[2.3 驗證 API 請求 3](#_Toc78196246)

[2.4 用python虛擬環境 3](#_Toc78196247)

[2.5建立一個App Engine應用 3](#_Toc78196248)

[2.6 建立雲端儲存 4](#_Toc78196249)

[2.7 執行應用 4](#_Toc78196250)

[2.7部署到App Engine 7](#_Toc78196251)

[2.8 app.yaml說明 8](#_Toc78196252)

[2.8.1 runtime\_config: python\_version: 3 9](#_Toc78196253)

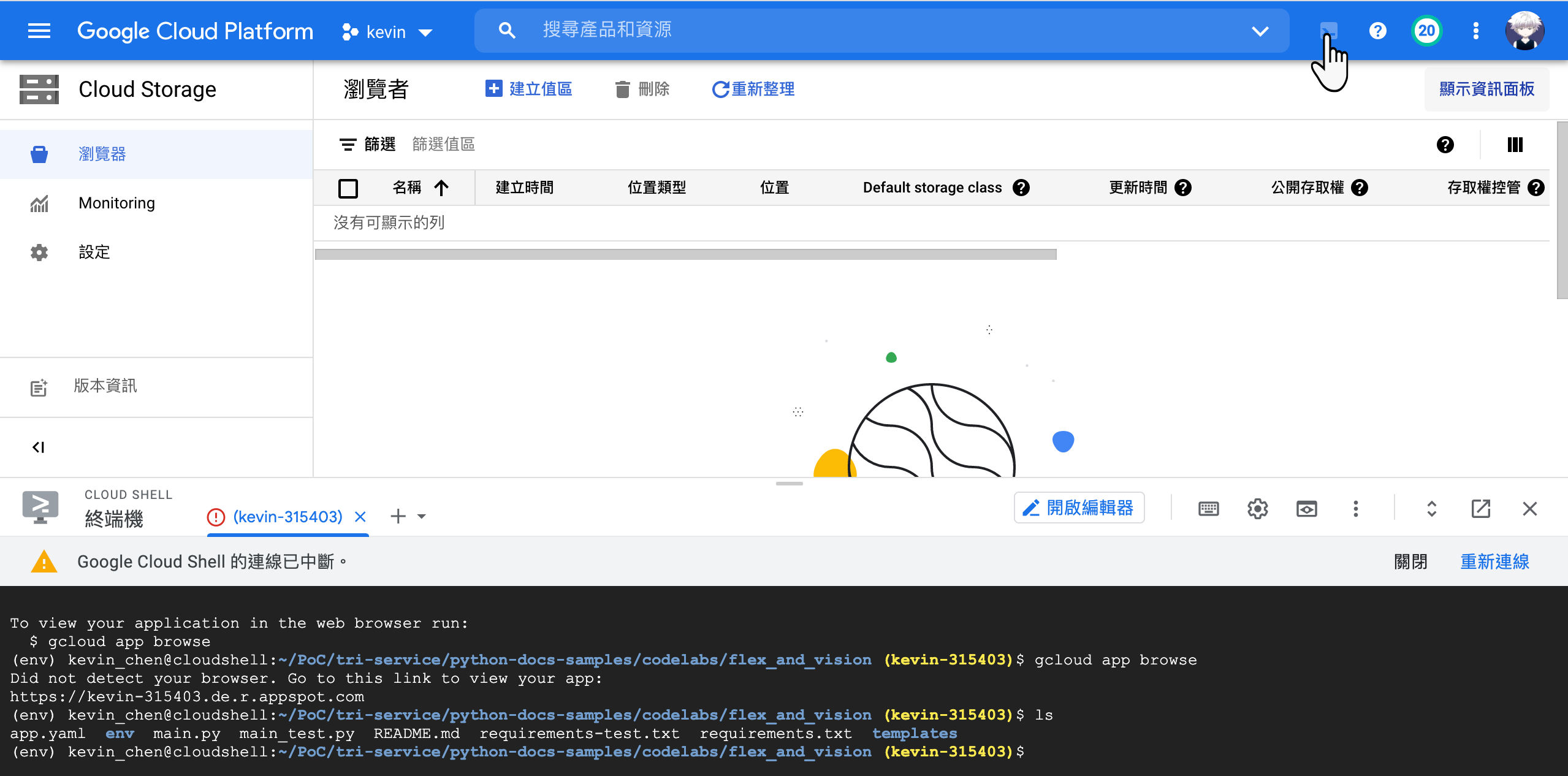
[2.8.2 env\_variables: CLOUD\_STORAGE\_BUCKET: <your-cloud-storage-bucket> 10](#_Toc78196254)

# 1. 前置條件

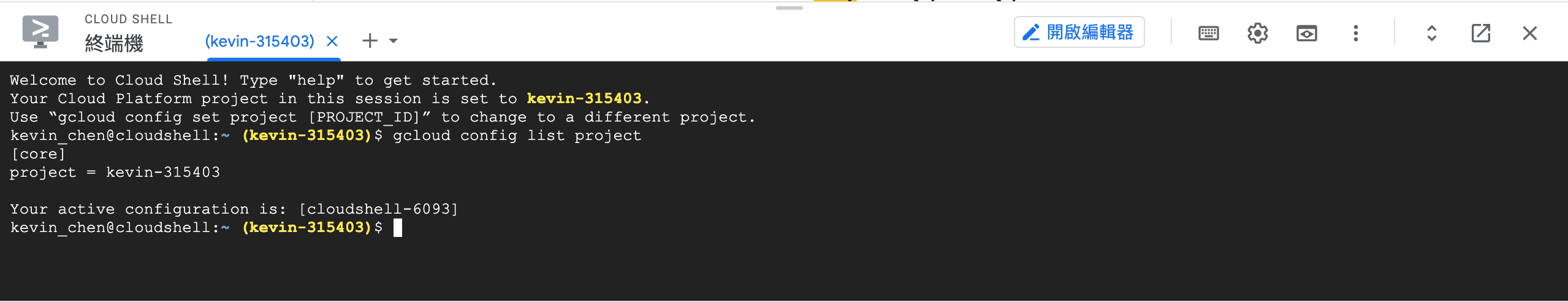
* 熟悉 Python
* 熟悉標準的Linux文本編輯器，如 vim、emacs 或納米
* 訪問帶臉的圖像

# 2.開始設定

登入Google Cloud Platform並進入Console畫面，點選右上角的「啟用Cloud Shell」，能開啟下方黑色底的cloud shell輸入介面。



## 2.1 在cloud shell輸入 gcloud config list project會顯示登入使用者可使用的project list



## 2.2 到Github下載您的python 模型訓練 repo ，在cloud shell CD使用者目錄後，輸入 git clone <https://github.com/YuehMintTai/RPython.git>

## 2.3 驗證 API 請求

在cloud shell 依序輸入以下命令，並修改合適的環境變數值

* 建立環境參數PROJECT\_ID，

export PROJECT\_ID=[YOUR\_PROJECT\_ID]

* 建立服務帳號存取Google Cloud APIs

gcloud iam service-accounts create rpython \ --display-name "RPython Service Account"

* 授予帳號權限

gcloud projects add-iam-policy-binding ${PROJECT\_ID} \ --member serviceAccount:rpython@${PROJECT\_ID}.iam.gserviceaccount.com \ --role roles/owner

* 建立服務帳號密鑰

gcloud iam service-accounts keys create ~/key.json \ --iam-account [rpython@${PROJECT\_ID}.iam.gserviceaccount.com](mailto:rpython@$%7bPROJECT_ID%7d.iam.gserviceaccount.com)

* 將服務帳號密鑰匯出做為環境變數

export GOOGLE\_APPLICATION\_CREDENTIALS="/home/${USER}/key.json"

## 2.4 用python虛擬環境

若不想用可以略過本節。

cd至rpython目錄下，依序在cloud shell執行下列命令列

* virtualenv -p python3 env
* source env/bin/activate
* pip install -r requirements.txt

將應用所需的python 模組，記錄在requirements.txt，便於管理。其格式如下範例：

Flask==2.0.1

gunicorn==20.0.4; python\_version > '3.0'

gunicorn==19.10.0; python\_version < '3.0'

google-cloud-vision==2.4.0

google-cloud-storage==1.38.0

google-cloud-datastore==2.1.4

## 2.5建立一個App Engine應用

* gcloud app create

## 2.6 建立雲端儲存

未來可以考慮將模型存放在bucket：

* export CLOUD\_STORAGE\_BUCKET=${PROJECT\_ID}
* gsutil mb gs://${PROJECT\_ID}
* 在合適的python程式片段，將scikit-learn模型存成檔案

dump(m, 'model.dmp')

ic = load('model.dmp')

confusion\_matrix(y\_test, ic.predict(X\_test))

* 將python建好的模型複製到bucket

BUCKET\_NAME = ‘<<PROJECT\_ID>>'

FILENAME = '訓練資料集'

DIRECTORY = ‘<<PROJECT\_ID>>/rpython’

STORAGE = os.path.join(DIRECTORY, FILENAME)

print(STORAGE)

#Download the datafile

#複製訓練資料集到bucket

!gsutil cp $FILENAME $DIRECTORY

model\_filename = 'model.dmp'

#複製模型到bucket

!gsutil cp $model\_filename $DIRECTORY

## 2.7 執行應用

* 可參考下列pythoon程式寫在main.py, 能執行Flask localhost httpd，執行方式在cloud shell裡執行 python main.py

from datetime import datetime

import logging

import os

from flask import Flask, redirect, render\_template, request

from google.cloud import datastore

from google.cloud import storage

from google.cloud import vision

CLOUD\_STORAGE\_BUCKET = os.environ.get("CLOUD\_STORAGE\_BUCKET")

app = Flask(\_\_name\_\_)

@app.route("/")

def homepage():

# Create a Cloud Datastore client.

datastore\_client = datastore.Client()

# Use the Cloud Datastore client to fetch information from Datastore about

# each photo.

query = datastore\_client.query(kind="Faces")

image\_entities = list(query.fetch())

# Return a Jinja2 HTML template and pass in image\_entities as a parameter.

return render\_template("homepage.html", image\_entities=image\_entities)

@app.route("/upload\_photo", methods=["GET", "POST"])

def upload\_photo():

photo = request.files["file"]

# Create a Cloud Storage client.

storage\_client = storage.Client()

# Get the bucket that the file will be uploaded to.

bucket = storage\_client.get\_bucket(CLOUD\_STORAGE\_BUCKET)

# Create a new blob and upload the file's content.

blob = bucket.blob(photo.filename)

blob.upload\_from\_string(photo.read(), content\_type=photo.content\_type)

# Make the blob publicly viewable.

blob.make\_public()

# Create a Cloud Vision client.

vision\_client = vision.ImageAnnotatorClient()

# Use the Cloud Vision client to detect a face for our image.

source\_uri = "gs://{}/{}".format(CLOUD\_STORAGE\_BUCKET, blob.name)

image = vision.Image(source=vision.ImageSource(gcs\_image\_uri=source\_uri))

faces = vision\_client.face\_detection(image=image).face\_annotations

# If a face is detected, save to Datastore the likelihood that the face

# displays 'joy,' as determined by Google's Machine Learning algorithm.

if len(faces) > 0:

face = faces[0]

# Convert the likelihood string.

likelihoods = [

"Unknown",

"Very Unlikely",

"Unlikely",

"Possible",

"Likely",

"Very Likely",

]

face\_joy = likelihoods[face.joy\_likelihood]

else:

face\_joy = "Unknown"

# Create a Cloud Datastore client.

datastore\_client = datastore.Client()

# Fetch the current date / time.

current\_datetime = datetime.now()

# The kind for the new entity.

kind = "Faces"

# The name/ID for the new entity.

name = blob.name

# Create the Cloud Datastore key for the new entity.

key = datastore\_client.key(kind, name)

# Construct the new entity using the key. Set dictionary values for entity

# keys blob\_name, storage\_public\_url, timestamp, and joy.

entity = datastore.Entity(key)

entity["blob\_name"] = blob.name

entity["image\_public\_url"] = blob.public\_url

entity["timestamp"] = current\_datetime

entity["joy"] = face\_joy

# Save the new entity to Datastore.

datastore\_client.put(entity)

# Redirect to the home page.

return redirect("/")

@app.errorhandler(500)

def server\_error(e):

logging.exception("An error occurred during a request.")

return (

"""

An internal error occurred: <pre>{}</pre>

See logs for full stacktrace.

""".format(

e

),

500,

)

if \_\_name\_\_ == "\_\_main\_\_":

# This is used when running locally. Gunicorn is used to run the

# application on Google App Engine. See entrypoint in app.yaml.

app.run(host="127.0.0.1", port=8080, debug=True)

* 在cloud shell右上角落可以點選預覽網站



## 2.7部署到App Engine

* 開啟編輯器修改app.yaml後儲存。

runtime: python

env: flex

entrypoint: gunicorn -b :$PORT main:app

runtime\_config:

python\_version: 3

env\_variables:

CLOUD\_STORAGE\_BUCKET: <your-cloud-storage-bucket>

* 執行部署，在cloud shell輸入 gcloud deploy app會需要5mins時間。

部署之後，輸入 gcloud browse app 系統將回應一個應用的網址。可以複製貼到瀏覽器查看。

## 2.8 app.yaml說明

.yml或.yaml都是ansible的文件。Ansible在各種電腦作業系統中，致力於IT資源管理自動化已有多時，舉凡：軟體安裝、部署、作業系統資源管理、網路管理。IT自動化的devops流程在實作，常需要ansible自動化安裝與部署的配置。

在cloud shell執行gcloud deploy app時，會呼叫Ansible的GCP管理模組協作完成安裝或部署工作。以下ㄒ小節說明app.yaml內的配置，詳細請參考

* GAE有兩種虛擬容器環境，視需要參考相應環境的app.yaml配置說明，本案例採用GAE彈性環境，以因應未來可能的效能需求配置：
  1. 標準環境（Standard Environment）：<https://cloud.google.com/appengine/docs/standard/python/config/appref>
  2. 彈性環境（Flexible Environment）：
     1. [https://cloud.google.com/appengine/docs/flexible/python/reference/app-yaml](file:///Users/KevinChen/Desktop/o%09https:/cloud.google.com/appengine/docs/flexible/python/reference/app-yaml)
     2. <https://cloud.google.com/appengine/docs/flexible/python/configuring-your-app-with-app-yaml>
* Ansible官方說明文件之Google Cloud Platform Guide：<https://docs.ansible.com/ansible/latest/scenario_guides/guide_gce.html>

下表為apply.yaml一般設定說明，上列B.1)官方文件還有cron.yaml, dispatch.yaml以及index.yaml配置說明。

Table 1：app.yaml一般設定

|  |  |
| --- | --- |
| **指令名稱** | **說明** |
| runtime: python | 必要設置。它是此應用程式使用的應用引擎語言執行時期的名稱。指定 Python會選擇 Python 2.7 或 3.7的完整實現。使用runtime\_config設置來選擇不同版本的 Python。此運行時間不支援任何應用引擎 API。您必須使用公開可用的 REST API 存取雲端服務。[閱讀更多](https://cloud.google.com/appengine/docs/flexible/python/runtime)  其他運行時間可用;有關詳細資訊，請參閱每種語言的文檔。 |
| env: flex | 選擇彈性環境 |
| entrypoint | 啟動應用程式的命令。入口點啟動回應環境變數PORT定義的埠上的 HTTP 請求的過程。有關更多資訊，請參閱[應用程式啟動](https://cloud.google.com/appengine/docs/flexible/python/runtime" \l "application_startup)。  註：在cloud shell: 打echo $PORT 可得知該環境變數值 |
| service: service\_name | 創建服務時需要。默認服務可選。每個服務和每個版本必須有一個名稱。名稱可以包含數位、字母和連字元。在靈活的環境中建議的服務名稱命名規則：**VERSION**-dot-**SERVICE**-dot-**PROJECT\_ID**  （其中VERSION是您的版本的名稱，SERVICE是您的服務的名稱，PROJECT\_ID是您的GCP專案 ID）的組合長度不能超過63個字元，不能以連字元開始或結束。為每個服務和每個版本選擇一個唯一的名稱。不要在服務和版本之間重複使用名稱。  註：SERVICE以前稱為"模組"。 |
| service\_account | 預覽-使用使用者管理的服務帳戶進行特定於版本的部署  自選。service\_account元素允許您指定使用者[管理的服務帳戶](https://cloud.google.com/appengine/docs/flexible/python/user-managed-service-accounts)作為版本的身份。在存取其他 Google 雲端服務和執行工作時，將使用指定的服務帳戶。  服務帳戶必須以下列格式提供：  service\_account: [SERVICE\_ACCOUNT\_NAME]@[PROJECT\_ID].iam.gserviceaccount.com |
| skip\_files | 自選。skip\_files元素指定了應用程式目錄中哪些檔不能上傳到應用引擎。值不是常規表達式，就是常規表達式清單。上傳應用程式時要上傳的檔案清單中省略了與任何常規表示式匹配的任何檔名。  例如，要跳過姓名以.bak結尾的檔，請添加以下skip\_files部分：  skip\_files: - ^.\*\.bak$ |

### 2.8.1 runtime\_config: python\_version: 3

指定Python版本，python\_version的有效值是：

* 3 使用最新支援的 Python 3.x 版本，目前為 3.7.2。
* 3.4 棄用。這使用 Python 3.4.8。
* 3.5 使用 Python 3.5.5。
* 3.6 使用 Python 3.6.8。
* 2 使用 Python 2.7.9 。

### 2.8.2 env\_variables: CLOUD\_STORAGE\_BUCKET: <your-cloud-storage-bucket>

定義環境變數，本立定義一個環境變數CLOUD\_STORAGE\_BUCKET，該變數直需要取自您在GCP定義一個App用的儲存容器；定義該儲存容器，可到GCP控制台完成配置：<https://console.cloud.google.com/storage/browser?authuser=1>

配置前，若您有多個GCP帳戶與多個專案，請確認進入的Cloud Storage環境是正確的，如下圖綠色匡標示。

