Developing Python applications with Google Cloud

Souvik Sarkar

September 1, 2023

Overview

This tutorial illustrates how to programmatically interact with Google Cloud Platform's (GCP) Storage and Vision APIs, using Python 3.x and other relevant tools on your local machine. Python developers can use the tutorial as a guide for the following purposes:

- Exploring and testing GCP's features and capabilities
- Integrating GCP products with their applications

The tutorial involves *automated detection of landmarks* in images of various locations, using the Google Cloud Storage and Vision APIs. You will learn how to:

- Configure various GCP products and services for your project.
- Set up the project with appropriate structure.
- Create a Python script for interacting with the APIs.
- Extend the project using various GCP offerings.
- Delete the project from GCP.

Note

- The commands and sample code are *unsuitable for production use*.
- All user-replaceable values are written in `UPPERCASE_CODE_FONT`.

Prerequisites

To maximize the learning experience, familiarity with the following is recommended:

- Development using Python 3.x
- Common Linux commands
- SDKs and libraries

Prior experience with cloud platforms is not necessary, but definitely helpful. The tutorial contains links to relevant documentation for each product.

Using GCP products

This tutorial uses the following GCP products within the free tier limits:

- **Google Cloud Storage**: Store assets and objects for your applications.
- **Google Vision API**: Use Google's pre-trained computer vision models for common image-based applications.
- **Google Cloud Client Libraries**: Develop application using client libraries for your chosen programming language.
- **Google Cloud SDK**: Deploy and manage applications using command-line tools.

Configuring google cloud

To set up your project, resources, and APIs in Google Cloud Platform:

1. Sign in to Google Cloud Platform using your Google Account credentials.
2. Create a new project and note the *Project ID*.
3. Create a storage bucket for the project and set the following *Permissions*:
   - Public access to `allUsers`
   - Access control to `Uniform`

   In addition, note the name of the bucket.

   Warning

   Setting public permissions exposes your bucket to unrestricted access. Avoid this in production environments.

4. Try the Vision API, then enable it for your project.

   Tip

   Do not generate authorization credentials when enabling the Vision API using the Google Cloud Console. In the next section, we create the authentication credentials using the `gcloud` utility.

Setting up the project

Create a development environment on your local machine with all necessary packages, Cloud SDK, and client libraries.

Creating a development environment

1. Ensure Python 3.x and `pip3` are installed on your system, and upgrade `pip` if necessary:

```
$ python3 --version
$ pip3 --version
$ pip3 install --upgrade pip
```

2. Create a project directory and navigate to it:

```
$ mkdir PROJECT_DIRECTORY
$ cd PROJECT_DIRECTORY
```

3. Set up a virtual environment:

```
$ pip3 install --upgrade virtualenv
$ virtualenv VIRTUALENV_NAME
$ source VIRTUALENV_NAME/bin/activate
```

Your terminal prompt will change to indicate you are within the virtual environment.

Installing google cloud sdk

1. Install the Cloud SDK for your operating system. Accept default options during installation.
2. Initialize the project:

```
$ gcloud init
```

When the program prompts, provide inputs that are relevant to the project.

Installing and configuring client libraries

1. Install the Storage and Vision client libraries for Python:

```
$ pip3 install --upgrade google-cloud-storage google-cloud-vision
```

2. Create a service account:

```
$ gcloud iam service-accounts create SERVICE_ACCOUNT_NAME
```

3. Grant permissions to the service account:

```
$ gcloud projects add-iam-policy-binding PROJECT_ID \
  --member="serviceAccount:SERVICE_ACCOUNT_NAME@PROJECT_ID.iam.gserviceaccount.com" \
  --role="roles/owner"
```

4. Generate a key file:

```
$ gcloud iam service-accounts keys create KEY_FILE.json \
  --iam-account=SERVICE_ACCOUNT_NAME@PROJECT_ID.iam.gserviceaccount.com
```

> **Warning**
>
> Ensure that the private key in the `KEY_FILE.json` file is unavailable to the public. If you use GitHub or similar services to host the code for this project, add the name of the key file to `.gitignore`.

5. Configure the authentication credentials by setting the `GOOGLE_APPLICATION_CREDENTIALS` environment variable to the name of the key file.

   - Append the following line to `~/.bashrc` (or your shell's equivalent):

   ```
   $ export GOOGLE_APPLICATION_CREDENTIALS="PATH/TO/PROJECT_DIRECTORY/KEY_FILE.json"
   ```

   - Source the file to make the changes effective:

   ```
   $ source ~/.bashrc
   ```

Creating a python script

Write a Python script that performs the following tasks:

1. Accepts a directory containing images of famous landmarks.
2. Uploads images to the Cloud Storage bucket.
3. For each uploaded image
    - Uses the Vision API to extract information about the landmarks present in the image.
    - Prints some information of common interest (for example, description and location of the landmark).

Project structure

To complete the project structure, create the following:

- A directory containing images of landmarks.
- A `main.py` file.

```
. PROJECT_DIRECTORY
|
|--- VIRTUALENV_NAME
|       |
|       |--- bin
|       |--- include
|       |--- lib
|       |--- lib64
|       |--- pyenv.cfg
|
|--- IMAGE_DIRECTORY
|       |
|       |--- image1.png
|       |--- image2.jpg
|       |--- image3.jpeg
|
|--- KEY_FILE.json
|
|--- main.py
```

Sample code in `main.py`

In the `main.py` file, write code which is similar to the following sample:

```
# Import dependencies
from __future__ import print_function
from google.cloud import storage, vision
import io, os

# Define functions
def get_image_names(image_dir):
    """Returns a list containing absolute paths of images."""
    image_abspath_list = []
    for file in os.listdir('./' + image_dir):
        image_abspath_list.append(os.path.abspath(image_dir + '/' + file))
    return image_abspath_list
```

```
def upload_landmark_images(bucket_name, image_abspath_list):
    """Returns a list containing relative URIs of uploaded images."""
    relative_storage_uris = []
    storage_client = storage.Client()
    bucket = storage_client.bucket(bucket_name)
    for image_abspath in image_abspath_list:
        image_name = image_abspath.split('/')[-1]
        blob = bucket.blob(image_name)
        blob.upload_from_filename(image_abspath)
        relative_storage_uris.append('gs://' + bucket_name + '/' + image_name)
    return relative_storage_uris

def get_landmark_information(relative_storage_uris):
    """Returns information on uploaded images."""
    vision_client = vision.ImageAnnotatorClient()
    image_object = vision.Image()
    for image_uri in relative_storage_uris:
        image_object.source.image_uri = image_uri
        vision_response = vision_client.landmark_detection(image=image_object)
        print('\n', '+' * 100, '\n')
        print('IMAGE:', image_uri, '\n')
        for landmark in vision_response.landmark_annotations:
            print('=' * 50)
            print('Landmark name:', landmark.description)
            print('Landmark location:', landmark.locations)
            print('Detection confidence score:', landmark.score)

# Accept user inputs
print('\n')
image_dir = input('Enter the image directory: ')
bucket_name = input('Enter the bucket name: ')

# Call functions
print('\n', 'Getting image names...')
image_abspath_list = get_image_names(image_dir)
print(' DONE', '\n')

print('\n', 'Uploading images to cloud storage...')
relative_storage_uris = upload_landmark_images(bucket_name, image_abspath_list)
print(' DONE', '\n')

print('\n', 'Extracting landmark information...')
get_landmark_information(relative_storage_uris)
print('\n', '+' * 100, '\n')
print(' DONE... All information displayed!')
print('\n\n')
```

Running the script

Run the `main.py` file. In the terminal, you will see an output similar to the following:

```
$ python3 main.py

Enter the image directory: IMAGE_DIRECTORY
Enter the bucket name: BUCKET_NAME

Getting image names...
```

```
DONE

Uploading images to cloud storage...
DONE

Extracting landmark information...
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

IMAGE: gs://BUCKET_NAME/IMAGE_1.jpg
================================================
Landmark name: Taj Mahal
Landmark location: [lat_lng {
  latitude: 27.174698469698683
  longitude: 78.042073
}
]
Detection confidence score: 0.8424403667449951
================================================
Landmark name: Taj Mahal Garden
Landmark location: [lat_lng {
  latitude: 27.1732425
  longitude: 78.0421396
}
]
Detection confidence score: 0.7699416875839233
================================================
Landmark name: Taj Mahal
Landmark location: [lat_lng {
  latitude: 27.166695
  longitude: 77.960958
}
]
Detection confidence score: 0.4865312874317169
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
DONE... All information displayed!
```

Extending the project

You can extend the project by using the following offerings from GCP:

- Use the Google Maps API to get the names of the landmark locations.
- Use Cloud SQL to store and retrieve image URLs.
- Use Google App Engine to deploy the application.

Deleting the project

If you discontinue development, delete the project from GCP to avoid incurring charges:

```
$ gcloud projects delete PROJECT_ID
```