Automating interaction with Cloud Storage and Vision API using Python

This tutorial describes how to automate the interaction between your local machine, Google Cloud Storage, and Google Cloud Vision. We use Python `3.x` as the programming language, and various tools offered by the Google Cloud Platform.

In this tutorial:

- About the Google Cloud Platform
- Configuring Google Cloud
- Setting up the project
- Creating a Python script
- Extending the project

About the Google Cloud Platform

Google Cloud Platform (**GCP**) offers Google's state-of-the-art computing infrastructure for various components of modern software development, such as:

- Virtual machines and containers
- Databases and object storage
- Application deployment and authentication
- Machine learning and computer vision
- Networking and security

This project uses the following GCP products (within the free usage limits):

- **Google Cloud Storage**: Store assets and objects for your applications.
- **Google Vision API**: Efficiently use Google's pre-trained computer vision models for common image-based applications.
- **Google Cloud SDK** and **client libraries**: Rapidly develop, deploy, and manage the application using command line tools.

Configuring Google Cloud

To configure the project, resources, and APIs in the 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 created in step 2, and set the following **Permissions**:
    a. **Public access** to `allUsers`
    b. **Access control** to `Uniform`

    In addition, note the name of the bucket.

    Caution: The permission settings expose the bucket and the objects within it to the public. Unrestricted public access reduces the complexity of the project, but is inappropriate for production environments.

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

    Note: When enabling the Vision API using the Google Cloud Console, don't generate any authorization credentials. In the next section, we generate the authentication credentials using the `gcloud` utility.

Setting up the project

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

Prerequisites

Ensure that Python `3.x` and `pip3` is available on your system.

```
$ python --version

$ pip --version

$ pip install --upgrade pip
```

Setting up a Python development environment

1. Create a project directory and navigate to it.

```
$ mkdir PROJECT_DIRECTORY

$ cd PROJECT_DIRECTORY
```

2. Create and activate a virtual environment.

```
$ pip install --upgrade virtualenv

$ virtualenv VIRTUALENV_NAME

$ source VIRTUALENV_NAME/bin/activate
```

   Notice that the prompt in your terminal changes to the following format:

```
(VIRTUALENV_NAME) [USER@HOST PROJECT_DIRECTORY]$
```

   The next sections of the tutorial assume that you are within the virtual environment.

Installing and configuring the Google Cloud SDK

1. Install the Cloud SDK for your machine's operating system. When the installation wizard prompts, accept the defaults by entering `y` throughout the installation process.

2. After completing the installation, initialize the project.

```
$ gcloud init
```

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

Installing the Storage and Vision client libraries

Install the Storage and Vision client libraries for Python `3.x`.

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

Configuring the Storage and Vision client libraries

1. Create a service account.

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

2. Grant permissions to the service account.

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

3. Generate the key file.

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

Warning: Ensure that the private key in the `.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`.

4. Configure the authentication credentials for your application code by setting the environment variable `GOOGLE_APPLICATION_CREDENTIALS` to the name of the key file.

```
$ export GOOGLE_APPLICATION_CREDENTIALS="KEY_FILE.json"
```

Note: The environment variable is set only for the current shell session. If you restart the shell, you must reset the environment variable.

Creating a Python script

In this section, we develop a Python 3.x script that automates the following:

1. Accept a directory containing images as an input. For this project, we use images of famous landmarks.
2. Upload the images to the Cloud Storage bucket created in the previous section.
3. For each uploaded image:
    a. Use the Vision API to extract information about the landmarks present in the image.
    b. Print some information of common interest (for example, description and location of the landmark).

Project structure

To complete the project structure, create the following:

1. A directory containing images of landmarks.
2. A file `main.py`.

Ensure that the project structure is similar to the following sample:

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

Code

In the file `main.py`, 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')
```

Output

Execute the file `main.py`. In the terminal, you should see an output similar to the following sample:

```
(VIRTUALENV_NAME) [USER@HOST PROJECT_DIRECTORY]$ python 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

In this project, you learned how to:

- Configure Google Cloud for your project
- Set up a Python development environment for cloud applications
- Programmatically use the Storage and Vision API

You can use these learnings and extend the project. Some ideas for further development are as follows:

- Use the Google Maps API to obtain the names of the landmark locations.
- Use Cloud SQL to store and retrieve the URLs of the images.
- Expose a REST API by creating a Flask application, and test it using Postman.
- Deploy the application to Google App Engine.
- Authenticate users using OAuth2.

If you discontinue development, delete the project to avoid incurring charges.

```
$ gcloud projects delete PROJECT_ID
```