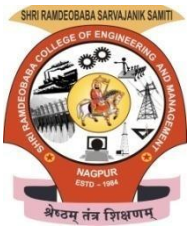

SHRI RAMDEOBABA COLLEGE OF ENGINEERING AND MANAGEMENT

Department of Computer Science & Engineering

Session: 2021-22

CLOUD COMPUTING (CST452-4)

ASSIGNMENT

VII Semester, B.E.

Shift-I

Group Members

Atul Thakre (42)

Nikhil Likhar (58)

Yash Agrawal (85)

Course Co-ordinator

Prof.Heena Agrawal

Table of Contents

Statement

Application of Google Drive Cloud APIs for syncing files.
To build a backup manager syncing files of local machine to Google Cloud.

Abstract

In this digital era, everything is going online, right from our home to big organizations. All paperwork has been transformed into digital documents. A lot of structured/unstructured data is being created in massive amounts and volume of such data is increasing exponentially. In these times, servers could crash, our systems could not handle such big data, and power failure could occur, all leading to loss of digital data.

Hence, it becomes necessary to backup our files and folders. Google Drive is a popular platform where people backup their files. But the task of backing up becomes time consuming as we have to upload each file and folder. Also, every time the contents of files are changed we need to delete it from Google workspace and upload again.

Therefore a Backup manager is helpful in such times which can sync our local machine with Google Cloud and takes the responsibility of backing up our files timely, making our job easy.

In this report we have tried to make such a Backup manager which syncs our file over Google Cloud using Google Drive API.

Tools& Languages Used

- Python
- Python Libraries: googleapiclient, os, shutil,pickle
- Google Drive API
- Google Cloud Console

Installation of Google Service:

1) We need to create an Outh2client in Google Cloud Console
2) Enable Google drive api by going in API & SERVICES in left panel
3) Create credentials for Outh2client
4) This will generate a JSON file having API secret key and secret credentials.
5) This needs to be downloaded in our Python Folder and create a service.
6) Whenever this service authenticates, it generates a token key which means we are fully authenticated to use Google Drive service.

Introduction

In present time people use multiple devices like phones, laptops, desktop PCs, smart watches, smart TVs, etc. There are multiple files like documents or media files that one needs to use across multiple devices. For instance you may need to work on a text file in your PC during night and then work on it back again from where you left, the next day on mobile. Transferring the file again and again over mail or using devices like pendrive for the purpose is quite cumbersome process.

Also, this is just one case, you may also require the same file over multiple devices. Multiple people may also need access to files and do changes to it whenever required. Cloud acts as a central place where all files from the local machine can be uploaded which can then be used by different devices. Files available in each device locally is also made available to other device connected to that cloud. Any changes to data like deletion of files, adding new files to local machine, etc are automatically reflected in cloud and hence in all the machines connected to the cloud.

This report demonstrates the working of above described syncer, which is built in python. The syncer is capable of reflecting any changes made to a specific directory or directories present on local machine to the cloud. During this practical execution different test cases like deleting files on local machine, deleting files on cloud, adding new files to cloud and adding new files to local machine were carried out. This execution helps one understand how cloud can be used to bring different devices close together and abstract the technical difficulties.

Implementation

Google drive cloud requires authentication to perform any operations, whether it is adding, deleting files or getting meta information. This provides inherent security to the cloud. Standard authentication practice of tokens is used in google cloud. A Tokens.pickle file is created which contains token information for authentication. If a tokens.pickle file is not present in the directory, user is asked to authenticate the app services, which creates a token.pickle file. If the file is already present, no authentication is required and token refresh is done automatically.

For each google cloud project, a credential is issued to developers which are used to ensure that APIs of google cloud are used by the owners of respective app only. These credentials need to be kept secure.

Google drive APIs are run with a standard services object , which is responsible for performing all operations in the cloud.

Some commonly used service object methods are:

Service.files().list()

Service.permissions().create()

Service.files().create()

When Files are added to local directory:

If a file is added in local directory, it will be automatically uploaded to the cloud when the script runs in next epoch. Those users who do not have this newly uploaded file in their local machine will automatically receive this file by the syncer's process.

When files are added to the cloud:

If certain files are added to the cloud directly without being present in any local machine, then those will automatically reflect in the local machines of all users connected to that cloud. Files can be added to cloud directly either manually or through other integrated systems.

When files are deleted from cloud but present in trash:

These files will be automatically deleted from the local machines of the user as they are no longer needed.

When files are deleted from cloud as well as trash:

The present implementation of syncer will upload a file to the cloud that is present in local machine, but is neither present neither in cloud nor in trash. However if meta data and some legacy information of cloud is maintained, then once a file is deleted from cloud, all local machines can be made to delete local version of that file as well.

Main Code for Project

```
import glob

import os

import shutil

from google.auth.transport import Response

import pandas as pd
```

```
import pickle
import re
import io
from googleapiclient.discovery import build
from google_auth_oauthlib.flow import InstalledAppFlow
from google.auth.transport.requests import Request
from googleapiclient.http import MediaIoBaseDownload
import requests
from tqdm import tqdm
import download_files
import upload_files

# If modifying these scopes, delete the file token.pickle.
SCOPES = ['https://www.googleapis.com/auth/drive.metadata',
        'https://www.googleapis.com/auth/drive',
        'https://www.googleapis.com/auth/drive.file'
        ]

def get_gdrive_service():
    creds = None
    # The file token.pickle stores the user's access and refresh tokens, and is
    # created automatically when the authorization flow completes for the first
    # time.
    if os.path.exists('token.pickle'):
        with open('token.pickle', 'rb') as token:
            creds = pickle.load(token)
    # If there are no (valid) credentials available, let the user log in.
    if not creds or not creds.valid:
        if creds and creds.expired and creds.refresh_token:
            creds.refresh(Request())
        else:
            flow = InstalledAppFlow.from_client_secrets_file(
                'credentials.json', SCOPES)
```

```
            creds = flow.run_local_server(port=0)
        # Save the credentials for the next run
        with open('token.pickle', 'wb') as token:
            pickle.dump(creds, token)
    # initiate Google Drive service API
    return build('drive', 'v3', credentials=creds)


def sync_files():
    cwd = os.getcwd()
    local_folder = cwd + "/local_folder"
    print("\n\nLocal Sync Folder : ",local_folder)
    # print(local_folder)
    local_files_list = os.listdir(local_folder)
    print("\n\nAll local files : ")

    for file in local_files_list:
        print(file)

    folder_metadata = {
        'name': "TestFolder",
        'mimeType': 'application/vnd.google-apps.folder'
    }

    print("\n\nGoogle Drive Sync Folder : ")
    service = get_gdrive_service()
    response = service.files().list(q="name='{name}' and
mimeType='{mimeType}'".format(name=folder_metadata['name'],
mimeType=folder_metadata['mimeType'])).execute()
    # print("\n\nResponse : \n",response)
    items = response.get('files', [])

    if not items:
        print("'{}' not found, create new".format(folder_metadata['name']))
```

```
    file = service.files().create(body=folder_metadata,
                                  fields='id').execute()
else:
    print("'{}'' found".format(folder_metadata['name']))
    file = items[0]

folder_id = file.get('id')
print("folderId={}".format(folder_id))

response = service.files().list(q="'{folderId}' in parents".format(folderId=folder_id)).execute()
# response = service.files().list().execute()
# print("\n\nGdrive Files response", response)
print("\n\n\nGdrive Files List : ")
gdrive_files_list = {}
for temp in response.get('files'):
    print(temp)
    gdrive_files_list[temp['name']] = temp['id']

# print("\n\n")
# for key, val in gdrive_files_list.items():
#     print(key, val)


trash_response = service.files().list(q="trashed = true").execute()
# print(trash_response)
print("\n\n\nTrash Files List :")
trash_files_list = {}
for temp in trash_response.get('files'):
    print(temp)
    trash_files_list[temp['name']] = temp['id']

print("\n\n")
# for key, val in trash_files_list.items():
```

```
    #     print(key, val)
    # print("\n\ngrive_files_list", gdrive_files_list.keys())
    # print("\n\ntrash_files_list", trash_files_list.keys())
    # print("\n\nlocal_file_list", local_files_list)

    # download

    print("\n\n----------  Downloading Syncing ----------\n")
    for file_name in gdrive_files_list.keys():

        # file_id = gdrive_files_list[file_name]
        # print("filename : ",file_name,"     Response :
",service.files().trash(fileId=file_id).execute())

        if file_name not in trash_files_list and file_name not in local_files_list :
            file_id = gdrive_files_list[file_name]
            # make it shareable
            service.permissions().create(body={"role": "reader", "type": "anyone"},
fileId=file_id).execute()
            # download file
            # print("\n\nlocal_folder", local_folder)
            # print("\n\nFile_name", file_name)
            download_files.download_file_from_google_drive(file_id, local_folder+"/"+file_name)
            print()
    print("-------------------------------------------------")

    # upload

    print("\n\n\n----------  Upload Syncing ----------")
    for file_name in local_files_list:
        if file_name in trash_files_list.keys():
            path = local_folder+"/"+file_name
            if os.path.exists(path):
```

```
            print("\nRemoving Local File : ", file_name)
            os.remove(path)

        elif file_name not in gdrive_files_list.keys():
            upload_files.upload_files(local_folder, file_name, folder_metadata['name'])

    print("\n-------------------------------------------------\n\n")

if __name__ == '__main__':
    sync_files()
```

Flow of Project

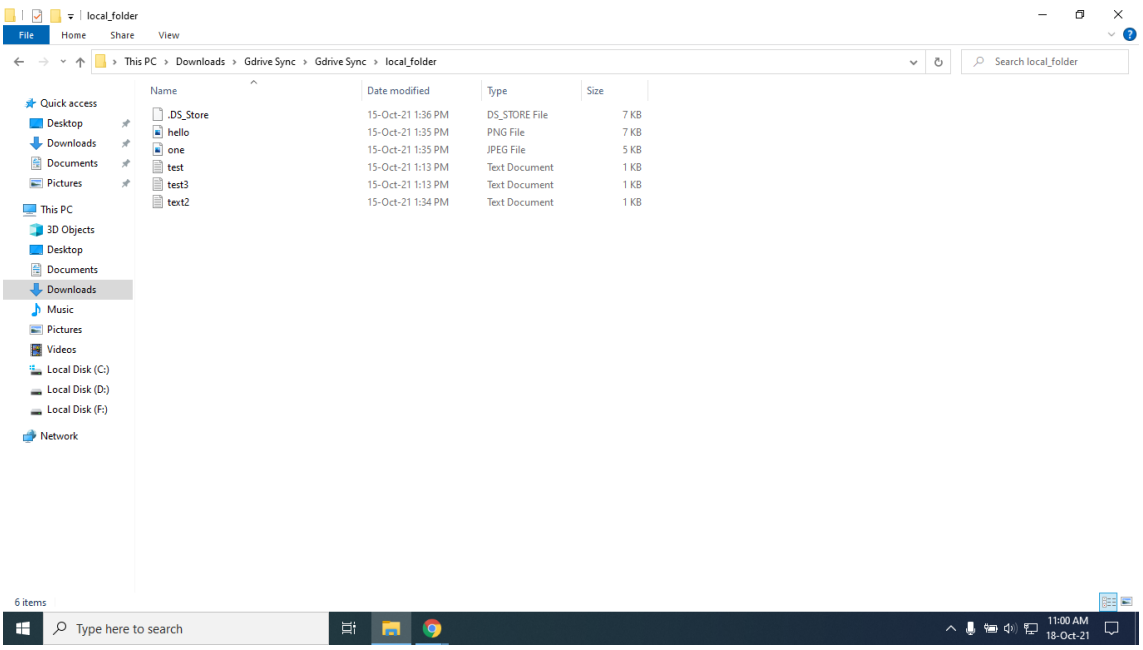

Figure 1: Folder to be synced

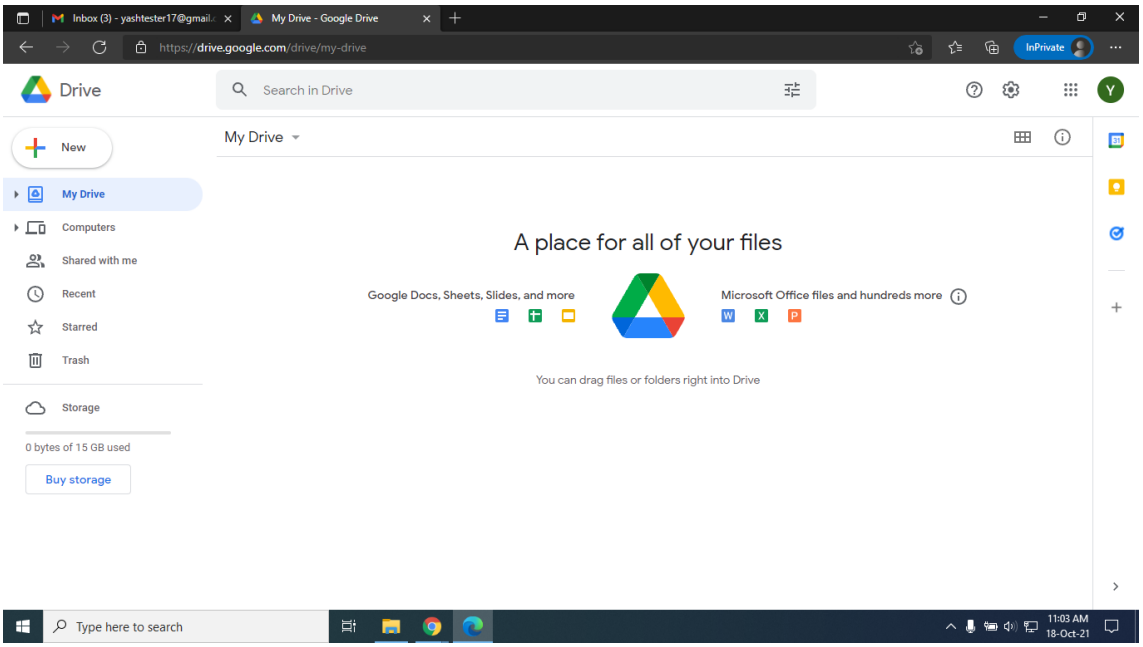

Figure 2: Google Workspace

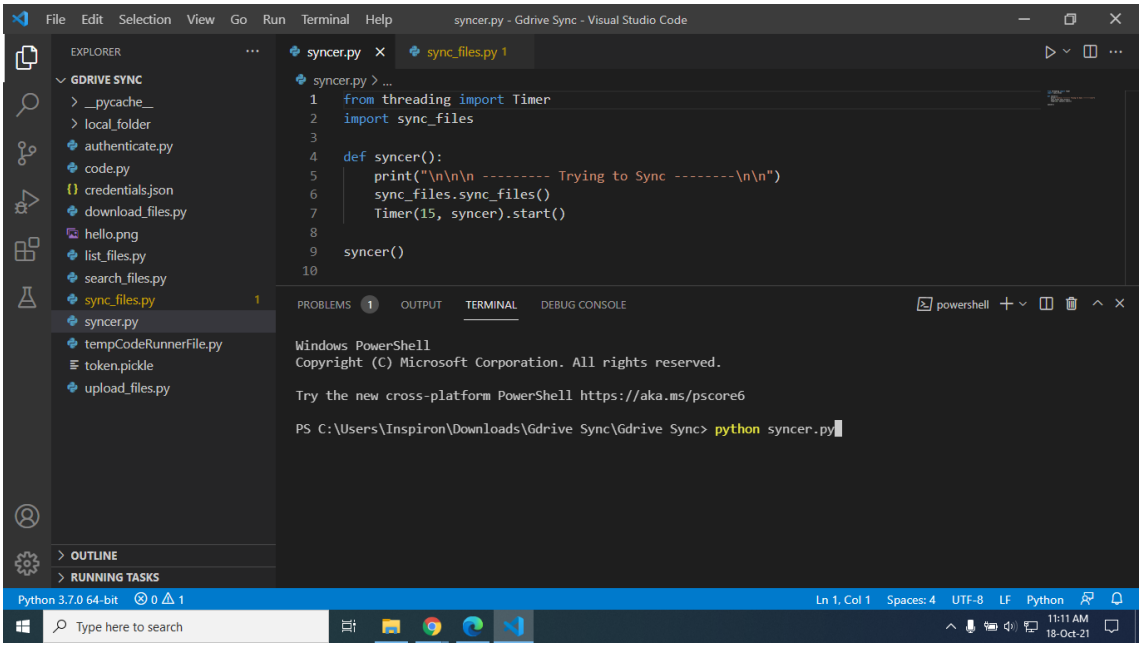

Figure 3: Running the Cloud Syncer

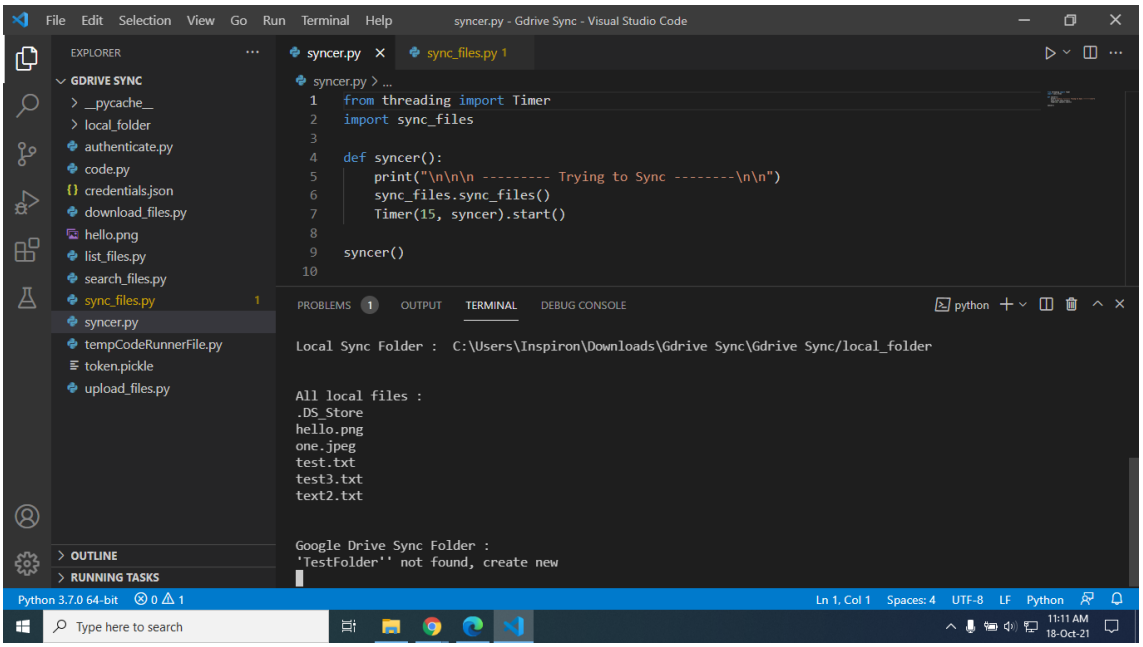

Figure 4: Detecting files of Local Machine

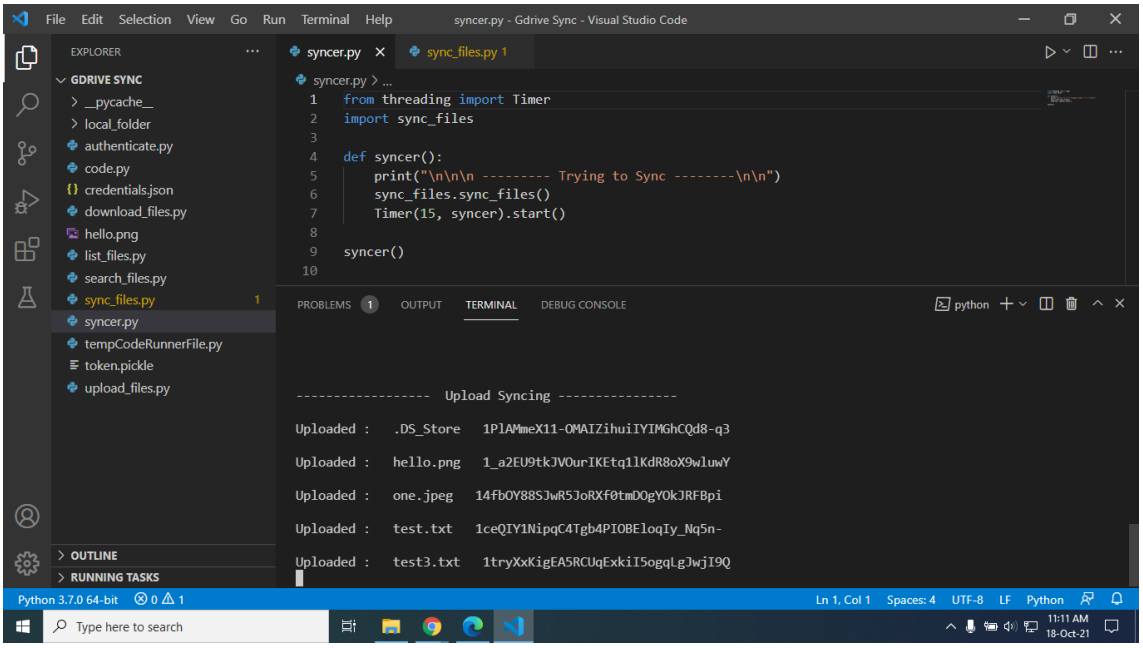

Figure 5: Uploading files from Local Machine to Cloud

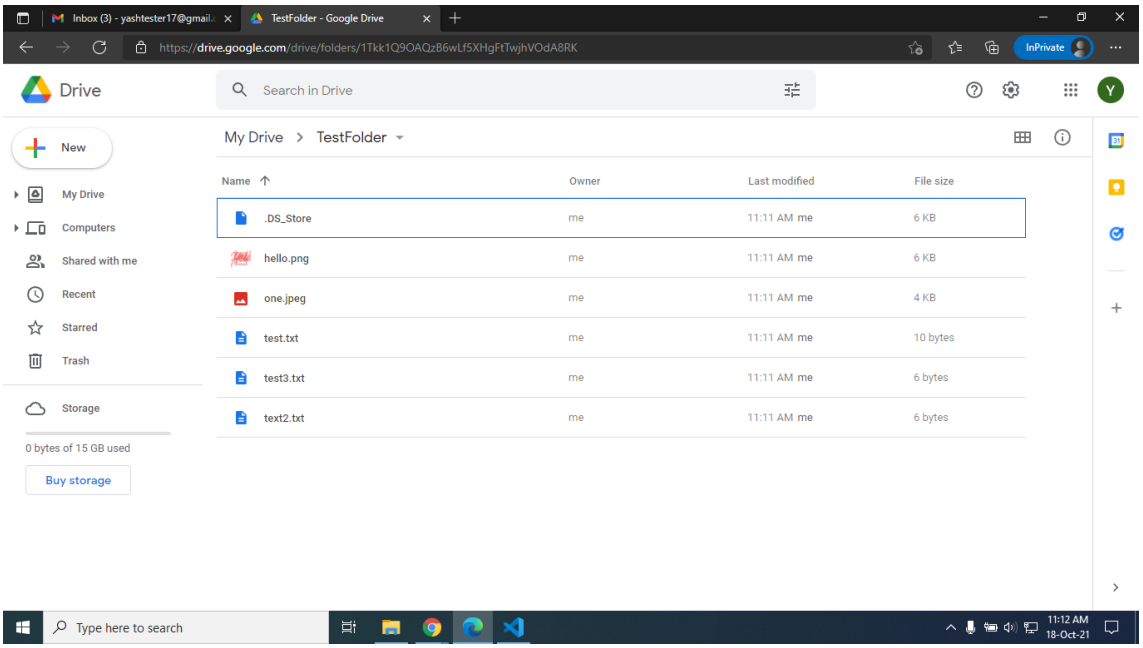

Figure 6: Local files synced with Cloud

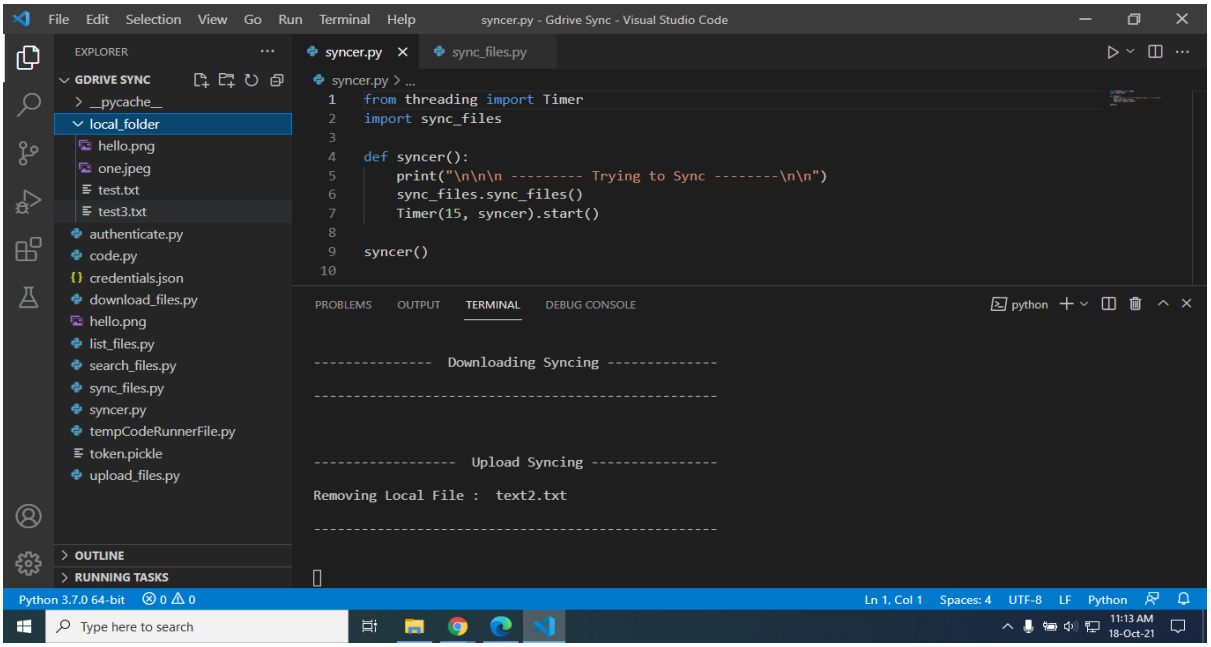

Figure 7: Changes in Cloud detected in Local Machine by Syncer

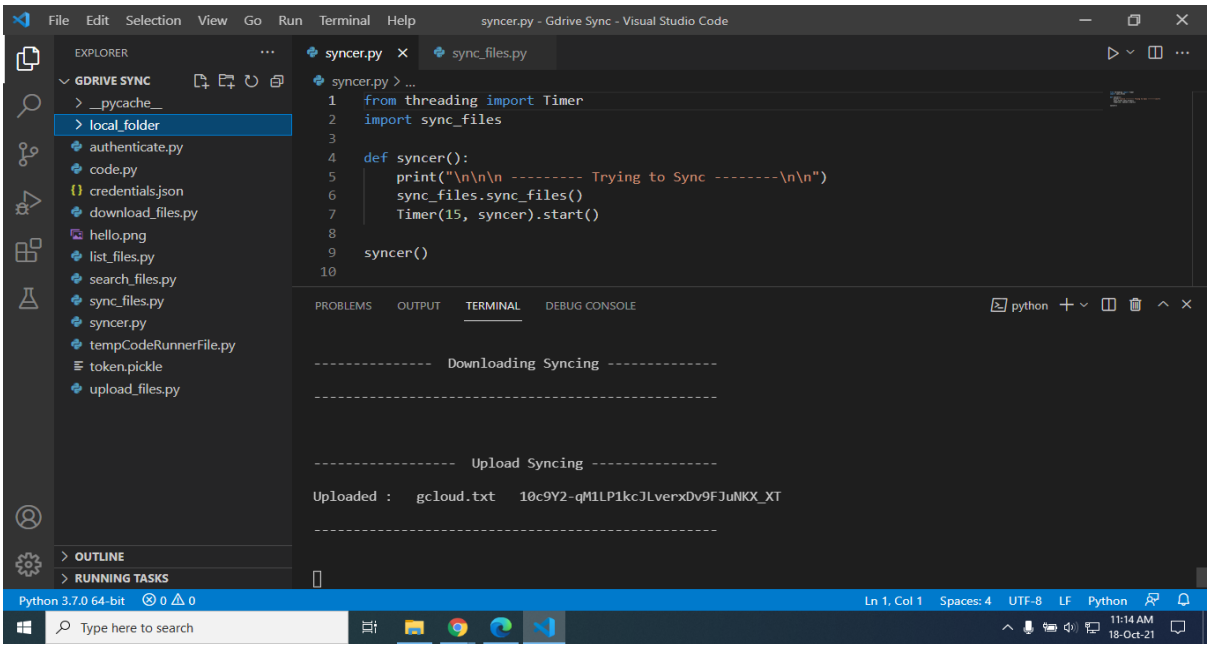

Figure 8: New files in Local Machine uploaded to the Cloud

References

1) Python Quickstart for Google Drive : https://developers.google.com/drive/api/v3/quickstart/python
2) Google Cloud: https://cloud.google.com/docs
3) YouTube Tutorial : https://youtu.be/9K2P2bWEd90
4) Google Cloud Console: https://console.cloud.google.com/