### Technical Document: UploadvideotoGDrive.py Script

## ****Introduction****

This technical document outlines the functionality, design, and operational assumptions behind the UploadvideotoGDrive.py script. This Python script is designed to automate the process of uploading video files stored in an SSD or any local storage device to a specific folder in Google Drive. The script runs at regular intervals (every 30 minutes by default), uploads all files in the specified folder, and then clears the uploaded files from the local directory to free up space.

## ****Purpose****

The primary purpose of the script is to ensure that video files generated by a CCTV or similar system, stored in the local SSD, are periodically uploaded to a predefined Google Drive folder. This helps mitigate the risk of local disk space overflow, ensures secure and accessible cloud storage of video files, and provides a scalable way to manage large volumes of video data.

## ****Assumptions****

· **File Format**:

* The script currently works with **MP4 format video files**. However, it can be extended to support other file formats by simply updating the file extensions in the code.

· **File Splitting**:

* The video files are **split by size or timestamp** in the SSD folder to prevent any single file from growing too large. This ensures efficient uploads and minimizes the risk of large file transfer interruptions.

· **Internet Connectivity**:

* The system assumes that the **internet connection is stable and strong**. The time taken for uploading is directly dependent on the strength of the internet connection and the file size.

· **Time for Upload**:

* The **upload time** varies based on the file size and internet bandwidth. Typically, large files or slow connections will increase the time taken to complete the upload.

· **Script Scheduling**:

* The script is **designed to run every 30 minutes** using Python's internal timer (sleep). However, in production environments, it's recommended to use **Windows Task Scheduler** (for Windows) or **Cron Jobs** (for Linux) to trigger the script at regular intervals. After the files are uploaded, the script deletes or moves the files from the SSD to avoid local storage issues.

· **Google Drive Configuration**:

* It is assumed that **enough space** is available on the Google Drive account, and proper credentials are provided via a cred.json file. This credentials file must be created by enabling Google Drive API access and setting up the OAuth 2.0 flow. The script also assumes that appropriate **test users** are created for API testing as needed.

## ****Flow****

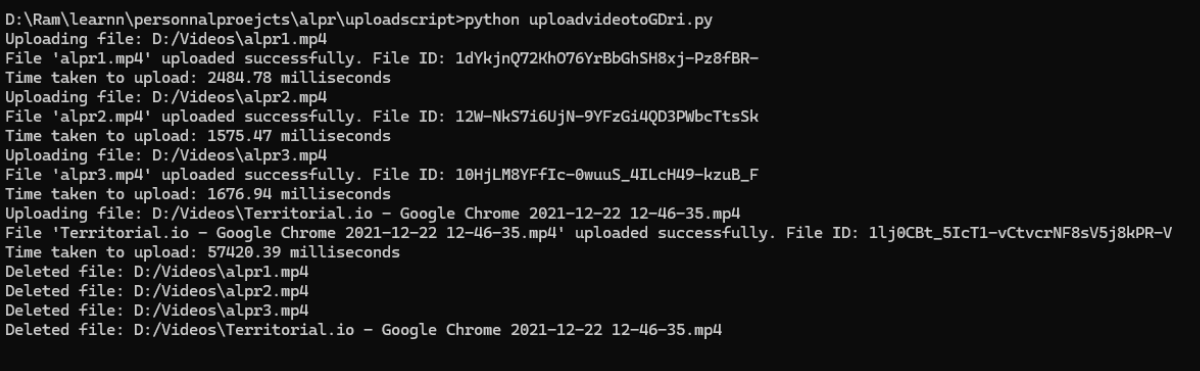
1. Video files are captured and stored in a local SSD.
2. Every 30 minutes, the script runs and checks for any MP4 files in the SSD directory.
3. If any files are found:
   1. They are uploaded to a **specific folder** on Google Drive (identified by the folder ID).
   2. Once successfully uploaded, the files are either **deleted** from the SSD to free up space or moved to an archive location.
4. The script waits for another 30 minutes before checking again.

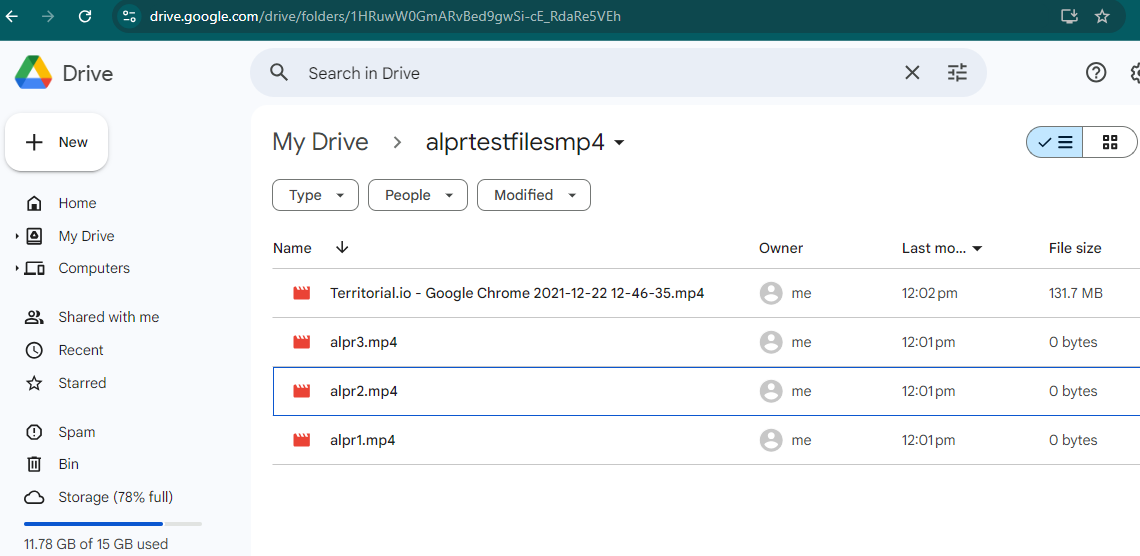
## ****Steps to Run the script****

1. Ensure python is installed in the server/desktop. Download [UploadvideotoGDrive](https://github.com/Ramkumar-stack/AALPR).py script and place in folder from where it can be executed
2. Ensure the ssd\_directory is specified to the right folder. GoogleDrive folder id is specified correctly in the python script. Ensure cred.json file is placed (refer download credentials below)
3. Run this to import dependent classes

#pip install --upgrade google-api-python-client google-auth-httplib2 google-auth-oauthlib

1. From the dos prompt where this script is placed along with Cred.json file (update the code of this location) run . “UploadvideotoGDrive.py’





## ****Google Drive set up prerequisities****

1. Ensure generic google Drive instead of personal google drive to avoid compromising passwords and other content in googledrive,gmail and photos etc.

**Step 1: Access Google Cloud Console**

1. Go to [Google Cloud Console.](https://console.cloud.google.com/)
2. If you haven’t done so already, sign in using your Google account.

**Step 2: Create or Select a Project**

1. In the top left corner, click the **Project** dropdown (usually next to the Google Cloud logo).
2. Select an existing project or click **New Project** to create a new one.
   * Name your project.
   * Choose an **organization** (optional).
   * Click **Create**.

**Step 3: Enable the Google Drive API (or any other required API)**

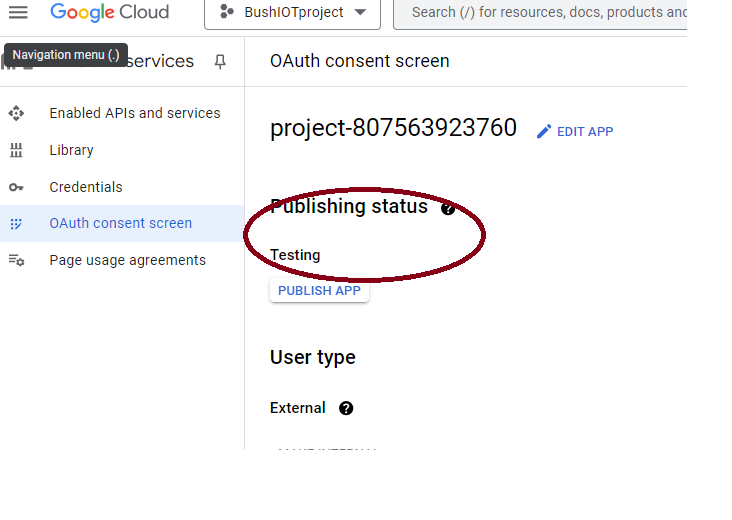
1. From the left-hand menu, click on **APIs & Services** → **Library**.
2. In the search bar, type **Google Drive API** (or the API you want to use).
3. Click on **Google Drive API** from the search results.
4. Click the **Enable** button to enable the API for your project.

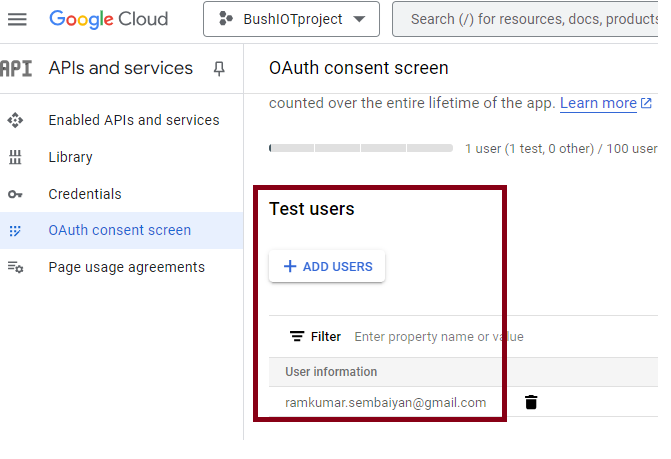
**Step 4: Create OAuth 2.0 Credentials**

1. After enabling the API, go to **APIs & Services** → **Credentials** from the left-hand menu.
2. Click the **Create Credentials** button at the top.
3. In the dropdown, select **OAuth 2.0 Client IDs**.

**Step 5: Configure the OAuth Consent Screen**

1. If you haven’t set up the **OAuth consent screen**, you’ll be prompted to do so.
   * Click on **OAuth Consent Screen** in the left menu.
   * Choose **External** (for most use cases) or **Internal** (if it’s only for users within your organization).
   * Fill in the required information such as **App Name**, **User Support Email**, and **Developer Contact Information**.
   * Click **Save and Continue** through the remaining screens. You don’t need to configure everything right now for a prototype.
   * Ensure ‘Publishing Status’ as ‘Testing’ and add a TestUser

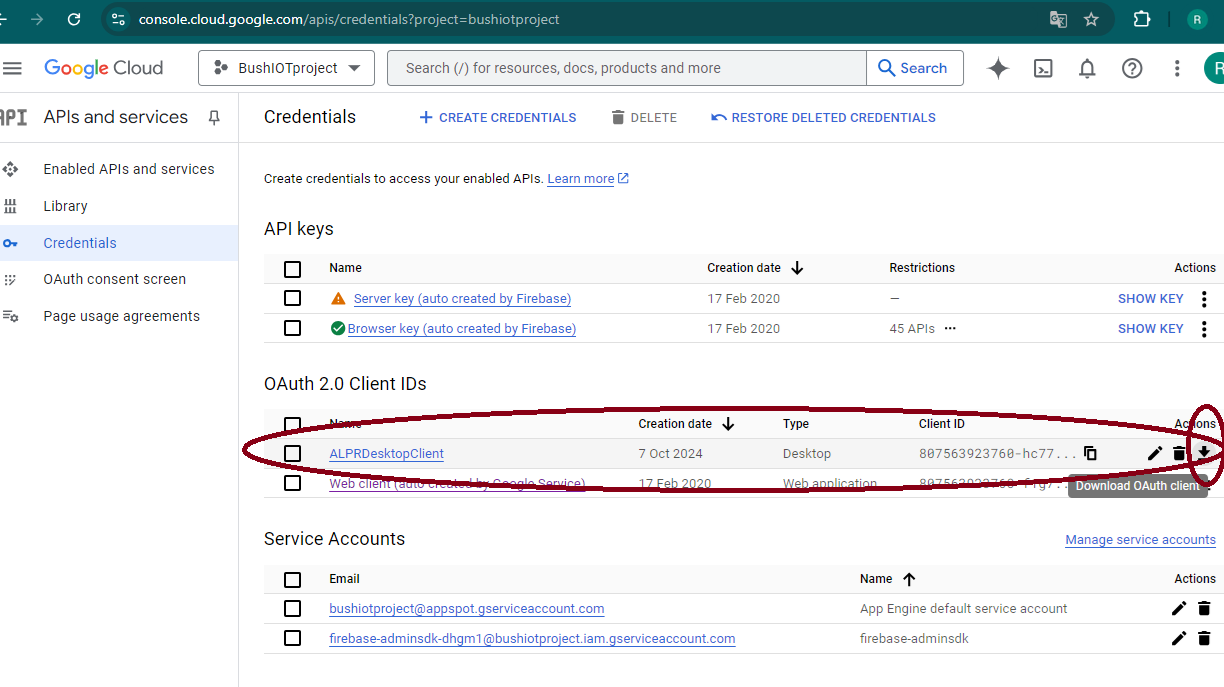




**Step 6: Create OAuth 2.0 Client ID for Desktop Application**

1. After configuring the consent screen, go back to **APIs & Services** → **Credentials**.
2. Click **Create Credentials** → **OAuth 2.0 Client IDs**.
3. In the **Application Type** dropdown, select **Desktop app**.
4. Enter a name for your OAuth client (e.g., **My Desktop App**).
5. Click **Create**.

**Step 7: Download Your Credentials**

1. Once created, you’ll see a **Client ID** and **Client Secret**.
2. Click **Download** to save the cred.json file to your computer. This file contains the OAuth 2.0 credentials (Client ID and Secret) that your application will use to authenticate.
3. 

**Step 8: Store Credentials Safely**

1. Keep your cred.json file secure; do not share it publicly, as it contains sensitive information (Client ID and Secret). This is referred in the python script to upload file to G: drive

**Configuring directory in GoogleDrive so that the files are upload to this specific folder instead of root directory**

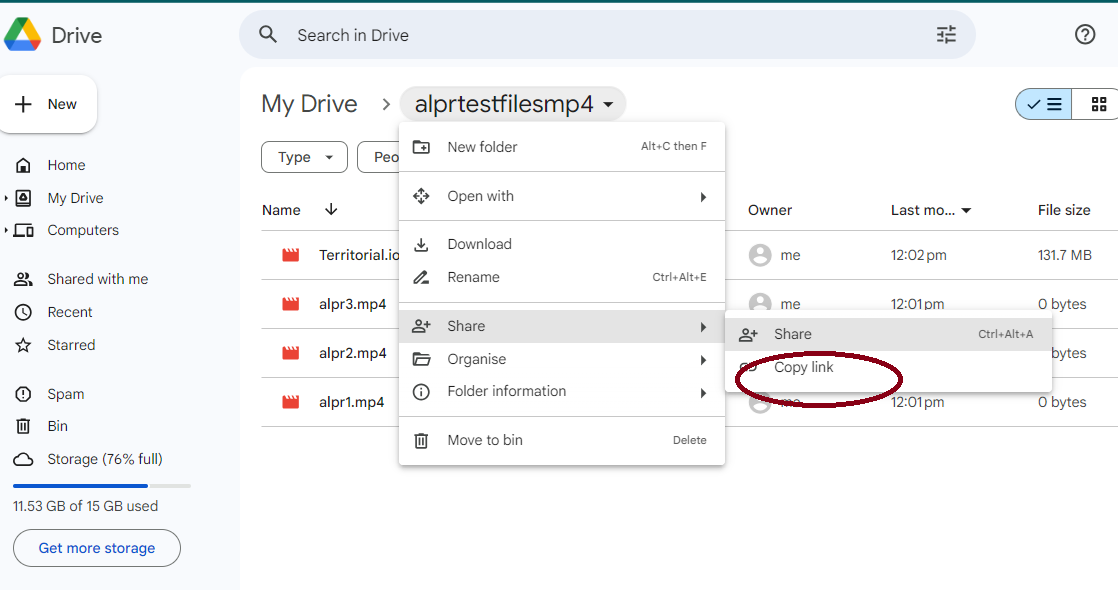
To upload a file to a **specific directory** (or folder) in Google Drive, you need to first find the folder's **ID** and then specify this ID when uploading the file.

Here's how to upload the file into a folder named alprtestfilesmp4:

### Steps:

**1.Find the Folder ID** of alprtestfilesmp4in Google Drive.

* 1. Go to your Google Drive, find the folder (alprtestfilesmp4), right-click on it, and click on **"copy link"**.

1. 
   1. In the link, you'll see something like this:

   #https://drive.google.com/drive/folders/1HRuwW0GmARvBed9gwSi-cE\_RdaRe5VEh?usp=drive\_link"

Use the highlighted digest in the ‘folder\_id’ variable in the script.

**How to make both ethernet and wifi ip works simultaneously in RaspPi.**

To use both an Ethernet-connected camera and Wi-Fi for internet access on a Raspberry Pi, you’ll need to configure both the Ethernet interface (for the camera) and the Wi-Fi interface (for internet access) separately. Assigning a static IP address to the Ethernet interface ensures that the camera connection remains stable while the Raspberry Pi uses Wi-Fi for internet access.

Here's how to achieve this:

### Step 1: Set a Static IP Address for the Ethernet Interface

sudo nano /etc/dhcpcd.conf

**Configure the Ethernet Interface (eth0)**: Add the following configuration at the end of the file:

*interface eth0static ip\_address=192.168.1.100/24 # Replace this with the desired static IP for the Ethernet connectionstatic routers=192.168.1.1 # Replace with your network's router IPstatic domain\_name\_servers=192.168.1.1*

Adjust 192.168.1.100 and 192.168.1.1 to your specific network configuration. This assigns the static IP address for the Ethernet-connected camera.

**Save and Exit**: Press CTRL + X, then Y, and Enter to save the file and exit.

### Step 2: Connect to Wi-Fi

**Configure Wi-Fi Connection**: Open the Wi-Fi configuration file:

sudo nano /etc/wpa\_supplicant/wpa\_supplicant.conf

network={

ssid="YourWiFiSSID"

psk="YourWiFiPassword"

}

### Step 3: Restart Networking Services

Restart the networking services for the changes to take effect:

sudo systemctl restart dhcpcd

### Step 4: Verify Connections

**Check Ethernet Connection**: Use the following command to check if the Ethernet interface (eth0) is connected and using the static IP address:

ifconfig eth0

Add “sudo dhclient” In the reboot script.

### Steps to Set Up a systemd Service for Auto-Restart

**Create a New Service File**:

Open a new service file in the /etc/systemd/system/ directory:

sudo nano /etc/systemd/system/my\_script.service

**Define the Service Configuration**:

Add the following content to configure your service. Replace /path/to/your/script.sh with the actual path to your shell script.

[Unit]

Description=My Script Service

After=network.target

[Service]

ExecStart=/path/to/your/script.sh

Restart=always # Automatically restart the service on failure

RestartSec=5 # Time in seconds to wait before restarting

User=pi # User to run the service under (use 'pi' if that's the Raspberry Pi user)

StandardOutput=journal # Log stdout to the journal

StandardError=journal # Log stderr to the journal

[Install]

WantedBy=multi-user.target

· **Save and Exit** the file by pressing Ctrl + X, then Y, and Enter.

· **Reload** systemd **to Apply the Service**:

·

**sudo systemctl daemon-reload**

**Enable the Service to Start on Boot**:

sudo systemctl enable my\_script.service

**Start the Service**:

sudo systemctl start my\_script.service

**Check the Status** (Optional):

You can check the status of the service to ensure it's running and restarting properly.

bash

Copy code

sudo systemctl status my\_script.service

Restart scritpt in raspberry Pi

*#! /bin/sh*

*# python -m venv myenv*

*. /home/iq-alpr/scripts/myenv/bin/activate  
  
echo 'welcome, to alpr'  
  
#pip install --upgrade google-api-python-client google-auth-httplib2 google-auth-oauthlib  
#pip install opencv-python  
  
#/home/iq-alpr/scripts/myenv/bin/python3 /home/iq-alpr/scripts/uploadtoGDriveNotimer.py  
  
#sudo dhclient to restart the client for network connection  
  
sudo dhclient  
  
/home/iq-alpr/scripts/myenv/bin/python3 /home/iq-alpr/scripts/SavecameraFeed.py >> /home/iq-alpr/scripts/logs/feeds.log*

*Script to record video from HikVision camera and save in SSD folder every 15 mins*

*import cv3  
import time  
import datetime  
import os  
  
import logging  
  
logging.basicConfig(filename='/home/iq-alpr/scripts/logs/alpr\_1.log', level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s')  
  
from uploadtoGDriveNotimer import main\_function1  
# IP camera URL (replace with the actual URL of your camera)  
#ip\_camera\_url = "http://<ip\_address>/video"  
ip\_camera\_url = "rtsp://[admin:mirage377@192.168.1.164](mailto:admin:mirage377@192.168.1.164" \t "https://mail.google.com/mail/u/0/" \l "sent/_blank)"  
  
# Video settings  
frame\_width =1920    # Set frame width (use the resolution that your camera supports)  
frame\_height = 1080  # Set frame height (use the resolution that your camera supports)  
fps = 25            # Frames per second  
video\_chunk\_duration = 15 \* 60  # Duration of each video chunk in seconds (30 minutes)  
  
def get\_filename():  
    """Generate a filename with timestamp for saving video files."""  
    #timestamp = datetime.datetime.now().strftime("%Y-%m-%d\_%H-%M-%S")  
    #return f"recording\_{timestamp}.mp4"  
    output\_dir="/home/iq-alpr/Videos"  
    timestamp = datetime.datetime.now().strftime("%Y-%m-%d\_%H-%M-%S")  
    return os.path.join(output\_dir, f"recording\_{timestamp}.mp4")  # Save to specified directory  
  
  
def main():  
    # Initialize video capture from the IP camera  
    cap = cv2.VideoCapture(ip\_camera\_url)  
  
    if not cap.isOpened():  
        [logging.info](http://logging.info/" \t "https://mail.google.com/mail/u/0/" \l "sent/_blank)("Error: Unable to connect to the camera.")  
        print("Error: Unable to connect to the camera.")  
        return  
  
    # Get default resolutions from the camera and set them  
    cap.set(cv2.CAP\_PROP\_FRAME\_WIDTH, frame\_width)  
    cap.set(cv2.CAP\_PROP\_FRAME\_HEIGHT, frame\_height)  
  
    # Define the codec and create VideoWriter object*

*fourcc = cv2.VideoWriter\_fourcc(\*'mp4v')  # Codec for MP4  
  
    while True:  
        print(' started Recording...')  
        [logging.info](http://logging.info/" \t "https://mail.google.com/mail/u/0/" \l "sent/_blank)(' started Recording...')  
        timestamp = datetime.datetime.now().strftime("%Y-%m-%d\_%H-%M-%S")  
        print(timestamp)  
        [logging.info](http://logging.info/" \t "https://mail.google.com/mail/u/0/" \l "sent/_blank)(timestamp)  
        # Create a new video file every 30 minutes  
        filename = get\_filename()  
        out = cv2.VideoWriter(filename, fourcc, fps, (frame\_width, frame\_height))  
  
        start\_time = time.time()  
  
        while (time.time() - start\_time) < video\_chunk\_duration:  
            ret, frame = cap.read()  
  
            if not ret:  
                print("Error: Failed to read frame from the camera.")  
                break  
  
            # Write the frame to the current video file  
            out.write(frame)  
  
        # Release the current video file after 30 minutes  
        out.release()  
        print(f"Saved: {filename}")  
        [logging.info](http://logging.info/" \t "https://mail.google.com/mail/u/0/" \l "sent/_blank)(f"Saved: {filename}")  
        print(' started Uploading...')  
        [logging.info](http://logging.info/" \t "https://mail.google.com/mail/u/0/" \l "sent/_blank)(' started Uploading...')  
        timestamp = datetime.datetime.now().strftime("%Y-%m-%d\_%H-%M-%S")  
        print(timestamp)  
        [logging.info](http://logging.info/" \t "https://mail.google.com/mail/u/0/" \l "sent/_blank)(timestamp)  
        main\_function1()  
        print(' Completed Uploading...')  
        [logging.info](http://logging.info/" \t "https://mail.google.com/mail/u/0/" \l "sent/_blank)(' Completed Uploading...')  
        timestamp = datetime.datetime.now().strftime("%Y-%m-%d\_%H-%M-%S")  
        print(timestamp)  
        [logging.info](http://logging.info/" \t "https://mail.google.com/mail/u/0/" \l "sent/_blank)(timestamp)  
  
  
    # Release the camera when done  
    cap.release()  
    cv2.destroyAllWindows()  
  
if \_\_name\_\_ == "\_\_main\_\_":  
    main()*