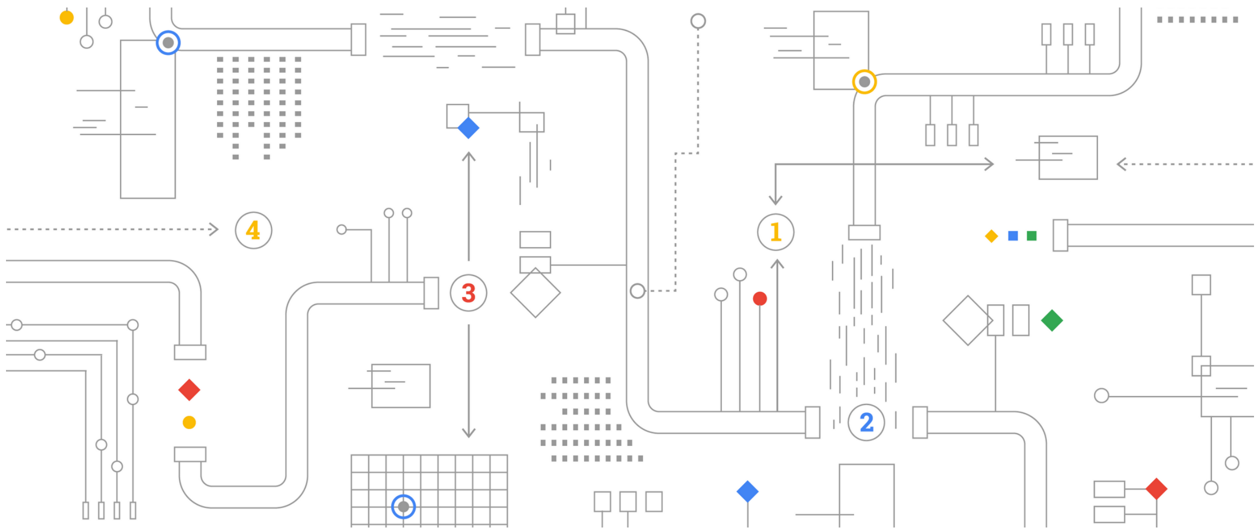


NOTE: This is not an official Google product. Please read the license at the end of this document

Table of Contents

Introduction

Customers require a user friendly tool that allows non-technical enterprise end-users to upload content to GCP. In media use cases, these uploads can be hundreds of gigabytes in size and are not suitable for upload via browser based interfaces.

The desktop uploader tool is designed for enterprises with non-technical end users who wish to upload large files from their desktops into the cloud on a regular basis. The desktop upload tool has the following features:

- Uploader desktop app
- Uses composable, parallel upload features of GCS
- Uses Google Auth in the cloud to prevent any propagation of sensitive credentials to the desktop app
- UI takes a list of files and folders in any combination and gives the user feedback on the progress of the various files

The system has been designed in a manner such that no sensitive information is shipped with the desktop app. All access to resources and credentials is provided through the Google authentication system. This allows the uploader to be a more secure desktop tool that can only be used by authorized users.

While the initial setup and configuration is a multi-step process for the system administrator, the actual operation of the uploader tool is, by design, exceedingly simple for the end user.

System Design

The system will consist of the following components:

1) A cloud based authenticator ("auth app")
2) A desktop based installable application that end users will use to upload data (desktop app)
3) An analytics ingest endpoint ("analytics app") in the cloud that receives updates from the desktop app and records various types of data/metadata about upload use

Authentication Overview

The desktop app cannot use the standard service account private key system to initialize the upload functionality, since new service account private keys will have to be created for each user and this will place an undue burden on the system administrator. Also, it lets sensitive data sit on a user's desktop (in the form of service account private keys) which can be a security risk.

The desktop app therefore uses a combination of in-memory refresh tokens and access tokens to initialize the GCS libraries. This technique can be used with either gsutil and the Java libraries. Since instrumentation and telemetry is not easily obtainable from gsutil, the desktop app currently uses the Java libraries for GCS, which are comparable to (and in some cases exceed) the performance of the Python-based gsutil utility.

Here is the flow for authentication:

1. User launches the desktop app
2. User clicks on the "Please Login" button, which opens a web page with a login button
3. User clicks the login button on the web page to sign in with Google authentication
4. An authentication popup appears that requests the user's credentials and shows them the list of permissions required
5. Once the user provides credentials and allows the permissions, Google auth responds to the web app with a "code" for offline access
6. The web application conveys this in code to the Auth App running in Cloud Run
7. The Auth app validates the offline token with Google's Authentication Backend, and retrieves user data corresponding to that token (email, name, etc.) and also obtains a refresh code and an access code.
8. The Auth app conveys the user information and the refresh + access codes back to the web client which then relays it back to the Desktop app (the desktop app functions like a local server)
9. The access code is only valid for 1 hr. When the user clicks the "Refresh Credentials" button on the Desktop app, or if the access token has less than 10 minutes of validity

remaining during an ongoing upload, the refresh token is sent back to the Auth app for fetching a new access code.

Source Code

To build from source, clone the following repository:
https://github.com/GoogleCloudPlatform/gcs-uploader

To build the code, you'll need Java 11 and Maven installed and functional

Preparation

Some initial preparation of the cloud environment is required

Permission Group

Create a Google group that can be used to control permissions into the GCS bucket(s) where uploads will be sent

Cloud Storage

Create the destination buckets, and assign the above created Google Group admin permissions into the buckets.

There are two types of buckets:

1. Destination Buckets: these buckets are where the uploaded content will be stored; these buckets have read/write permissions for the permissioned group created above.
2. Configuration Bucket: this bucket will store the configuration file; this bucket should NOT be shared with end users; the service account of the authentication app should have access to this bucket.

The destination buckets are listed in the configuration file detailed in the Authentication App section.

The configuration bucket contains the configuration file outlined in the Authentication App section. The Authentication app reads the contents of the configuration file and conveys it to the successfully logged in user.

OAuth Credentials

Create application OAuth credentials form the IAM area in Google Cloud Console. Ensure that `http://localhost:8080` is an authorized domain for the credentials.

Pub/Sub

Create the pubsub topic that will be used to send file upload notifications.
The enterprise administrator is responsible for processing the events that are transmitted to the the upload notification topic. Keep in mind that PubSub has an “at least once”

BigQuery

Create the BigQuery dataset and table. The table should have the following schema:

Field name	**Type**	**Mode**
guid	STRING	REQUIRED
type	STRING	NULLABLE
timestamp	TIMESTAMP	REQUIRED
index	INTEGER	NULLABLE
username	STRING	NULLABLE
status	STRING	NULLABLE
event	STRING	NULLABLE
blobId	STRING	NULLABLE
startTime	TIMESTAMP	NULLABLE
endTime	TIMESTAMP	NULLABLE
fileName	STRING	NULLABLE
size	INTEGER	NULLABLE
mediaId	STRING	NULLABLE
mbps	INTEGER	NULLABLE
timeTaken	INTEGER	NULLABLE
comment	STRING	NULLABLE

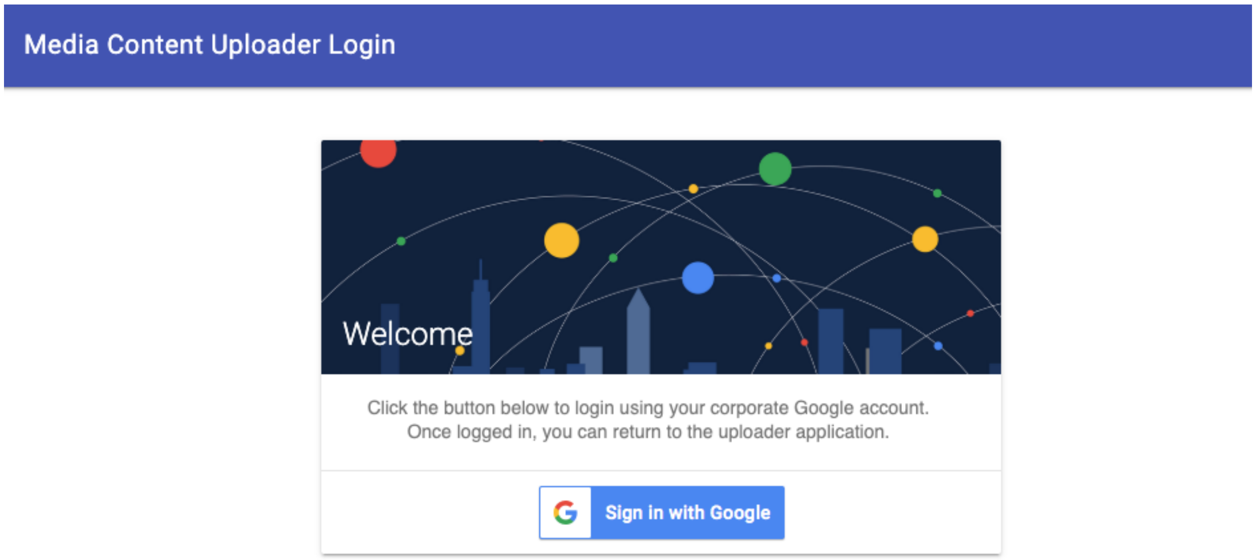
Components and Deployment

Cloud Authenticator ("Auth app")

The cloud authenticator is a Spring Boot based containerized app running in Cloud Run. The cloud authenticator performs the following functions:

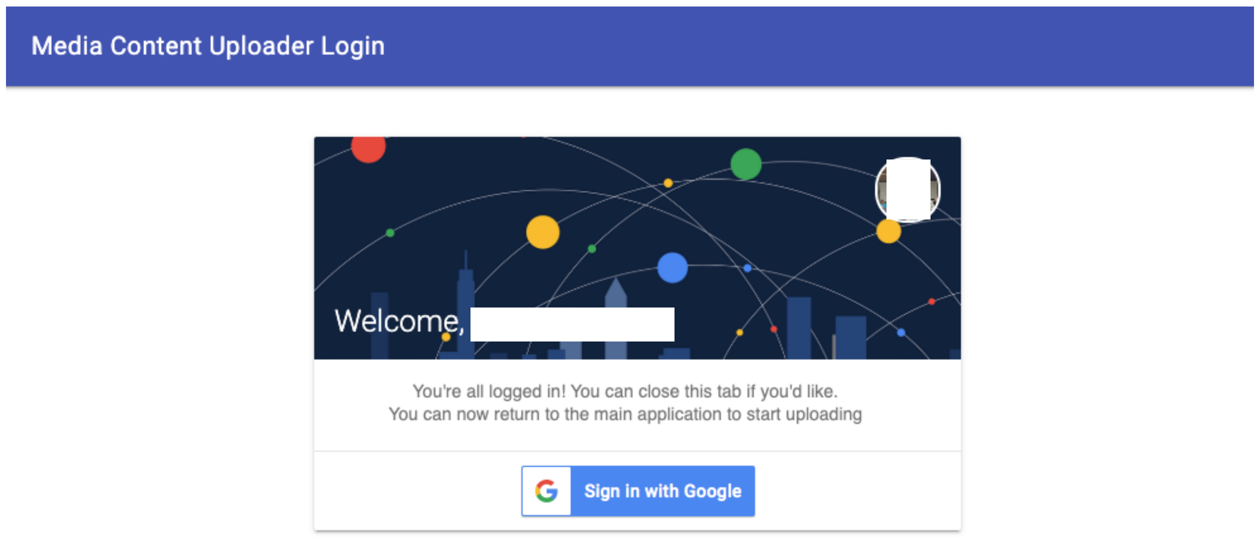
1. User Validation: when the user first signs into the Desktop app, the code provided by the user is validated with Google and user information, refresh code, and an access code is obtained, and sent back to the Desktop app.
2. Token refresh: when a refresh token is provided by the Desktop app, the Auth app checks the validity of the token with the Google Authentication backend and issues a new access token valid for another hour.

Authentication screen before login:



Click the button below to login using your corporate Google account. Once logged in, you can return to the uploader application.

Authentication screen after login:


Media Content Uploader Login

Welcome, [redacted]

You're all logged in! You can close this tab if you'd like.
You can now return to the main application to start uploading

When authentication is successful, the Auth App returns data from a configuration file regarding the various destinations and other metadata. The information required in this file is given in the following section.

Auth App Configuration

The Auth app requires the following environment variables to be configured:

- **DEST_CONFIG_BUCKET**: the bucket where the configuration file to be supplied to the users is stored
- **DEST_CONFIG_FILE**: the path and name of the config file in the DEST_CONFIG_BUCKET
- **CLIENT_ID**: the client id for the OAuth screen, obtained from the Credentials page in Google Cloud Console
- **CLIENT_SECRET**: the client secret associated with the client id
- **SCOPES**: the scopes for the OAuth process, must be set to:

```
profile email https://www.googleapis.com/auth/devstorage.read_write
```

  (note the spaces between the words)

DEST_CONFIG_FILE template (json file):

```
{"destinations":[
  {
    "gcsBucket":"<<dest-bucket-1>>",
    "gcsFolder":"open-encode",
    "displayName":"Open Encode"
  },
  {
    "gcsBucket":"<<dest-bucket-2>>",
    "gcsFolder":"dailies",
    "displayName":"Dailies Folder"
  },
  {
    "gcsBucket":"content-uploader",
    "gcsFolder":"raw",
    "displayName":"Raw Cuts"
  }
],
"analyticsEndpoint":"https://<<service-name>>.a.run.app",
"analyticsAuthUsername":"<<uuid>>",
"analyticsAuthPassword":"<<uuid>>"
}
```

`analyticsAuthUsername` should be set to be the same as the **SPRING_SECURITY_USER_NAME** in the analytics app in the next section

`analyticsAuthPassword` should be set to be the same as the **SPRING_SECURITY_USER_PASSWORD** in the analytics app in the next section

Building the App

In the `content-uploader` directory, switch into the `funct-auth` directory.
Build the application using Maven with the following command:

```
mvn clean package jib:build \
     -DskipTests \
     -DPROJECT_ID=<project_id> \
     -DIMAGE_NAME=<image_name>
```

Where `project_id` is the name of the project where the image will be deposited in the `gcr.io` repository, and `image_name` is the name of the image.

Deploying the App

In Cloud Run, create a new deployment, and select the image that was just created.
Add the environment variables as required
Start serving the version
Note the serving url of the Cloud Run instance, this will be required for the deployment of the Desktop App.

Cloud Analytics ("Analytics app")

The Analytics app is a secure endpoint that requires credentials provided by the Auth app for access. The Desktop app continuously sends analytics information to the Analytics app, which then inserts that data into the defined BigQuery dataset+table.

The following information is recorded by the Analytics app:

1. Logins by users
2. Uploads of files that occur without composite uploads ("single files"); the start time, end time, size, and average mbps are recorded.
3. For composite uploads, the start time of the root task (the base file), and start/end/time/mbps of each segment, and the final time taken for the full upload and stitching of the composite file are recorded.

The analytics app also fires an event into a Pub/Sub queue with the final "completion" event is received about a file (whether single or composite). This allows any downstream processors for the content to be activated on the content. The regular GCS upload triggers cannot be used because they will fire in the case of each uploaded segment of the composite upload, and this is not a useful trigger to act upon; downstream consumers only care about the final, fully composed file.

The data in the Analytics BigQuery dataset allows the uploader administrators to get insights as to how much content is being uploaded by each user on a daily basis, various statistics about average throughput and file sizes, etc.

Analytics App Configuration

The Analytics App requires the following environment variables to be configured:

- **UPLOAD_NOTIFICATION_TOPIC**: the Pub/Sub topic to which notifications of new files will be published
- **ENABLE_BIG_QUERY**: If true, then analytics data will be written to BQ. If false, the data received will not be written to BQ; the pub/sub notification events will still be generated. Defaults to 'true'. If set to 'false', then the following fields of 'DATASET_NAME' and 'TABLE_NAME' can be omitted (and if set, will be ignored)
- **DATASET_NAME**: the BigQuery dataset to which analytics data will be written; optional if 'ENABLE_BIG_QUERY' is 'false'
- **TABLE_NAME**: the BigQuery table in the dataset which will be used to record the actual analytics data; optional if 'ENABLE_BIG_QUERY' is 'false'
- **SPRING_SECURITY_USER_NAME**: an long, random username; you can use 'uuidgen' to generate a uuid to use for this setting
- **SPRING_SECURITY_USER_PASSWORD**: an long, random password; you can use 'uuidgen' to generate a uuid to use for this setting

Notifications in Pub/Sub

GCS normally offers an object notification: i.e. when an object is uploaded into GCS, it will automatically issue a PubSub notification. However, in this upload case, that is not desirable since GCS will issue notifications for each file segment as it is uploaded. The system should generate a notification only when the complete composite object is available, and not notify on the individual segments.

Therefore, when the Analytics App receives a “COMPLETE” notification event from the client, it will trigger a message in Pub/Sub notifying of the availability of the single or composite file. The notification will be a JSON object and will contain the following information:

- **`bucket`**: the name of the destination bucket where the file was uploaded
- **`objectName`**: the name of the object in the bucket created for the file
- **`fileName`**: the file name that was uploaded
- **`username`**: the email address of the logged in user that performed the upload
- **`mediaId`**: any media id that the user assigned the object on upload
- **`size`**: the size of the uploaded file, in bytes
- **`guid`**: the guid of the object, can be used to determine if this event is a re-issue

The sys admin must set up a system to accept this message for down stream process, based on enterprise requirements.

Building the App

In the `content-uploader` directory, switch into the `funct-analytics` directory.
Build the application using Maven with the following command:

```
mvn clean package jib:build \
      -DskipTests \
      -DPROJECT_ID=<project_id> \
      -DIMAGE_NAME=<image_name>
```

Where `project_id` is the name of the project where the image will be deposited in the `gcr.io` repository, and `image_name` is the name of the image.

Deploying the App

In Cloud Run, create a new deployment, and select the image that was just created.
Add the environment variables as required
Start serving the version
Note the serving url, and the username and password assigned to the Cloud Run instance, this will be required for the deployment of the file that will be served by the Auth App.

Config File

The configuration file is put in a separate bucket where it is served to authenticated users. The config file contains information about the upload destinations and also the address, username, and password of the Analytics app.

The config file is a json file with the following information:

- Array of destinations; each destination has the following fields:
    - **gcsBucket**: the destination bucket
    - **gcsFolder**: the folder in which the object will be deposited
    - **displayName**: the name that will be shown to the user
- Access information for the analytics endpoint:
    - **analyticsEndpoint**: the url of the endpoint
    - **analyticsAuthUsername**: the username created in the previous step
    - **analyticsAuthPassword**: the password created in the previous step

Here is a sample of a config file:

```
{
  "destinations":[
    {
      "gcsBucket":"content-uploader-bucket-1",
      "gcsFolder":"dailies",
      "displayName":"Dailies Folder"
    },
    {
      "gcsBucket":"content-uploader-bucket-2",
      "gcsFolder":"raw",
      "displayName":"Raw Cuts"
    }
  ],
  "analyticsEndpoint":"<provided_cloud_run_url>",
  "analyticsAuthUsername":"<uuid_username>",
  "analyticsAuthPassword":"<uuid_password>"
}
```

You can save a config file similar to the above in a bucket accessible by the application e.g.
`gs://<content-uploader-config-bucket>/uploader-config.json`

In the above config file, the user will see “Dailies Folder” and “Raw Cuts” as options for upload destinations. The files created in these destinations will be as follows: a file called “movie1.mp4” with destination “Dailies Folder” will be uploaded to the folder in the form:
`gs://content-uploader-bucket-1/dailies/yyyy/MM/dd/movie1.mp4`

Where yyyy, MM, dd are the 4 digit year, 2 digit month, and 2 digit day of the upload.

Desktop Application (“Desktop App”)

Features:

- Will launch a web page that will allow the user to click on Google Sign-In to get an OAuth token
- Will transmit this token to the server for validation and exchange it with a credential to be used with the java libraries and the GCS java client
- End Users will be able to drag and drop files and folders onto the desktop interface
- The app will then trigger gsutil with the right set of commands, using the authentication token provided by the Cloud Authenticator system
- The app captures progress information from the ongoing upload process and updates the user interface to reflect the progress and throughput being achieved for each file

The desktop application can be packaged as separate zip files for mac and windows installations.

The zip file can contain the full application that will allow users to launch the app by double clicking on an icon or executable or a batch/shell file; the app is a java executable so will require the java runtime to be packaged along with it.

The desktop app requires the user to enter a media id for each file being uploaded. The user must hit ‘enter’ or ‘tab’ after each entry so that it will be accepted by the system. Files that do not have a media id will not be uploaded.

Building the Application

In the `content-uploader` directory, switch into the `app-desktop` directory.
Build the application using Maven with the following command:

```
mvn clean package
```

This will create a file called `cu.jar` in the `target/` folder.
This jar file is a fully self-contained application.

Deploying the application

Deliver the jar file to end user’s desktops via various desktop control and automation tools.
The users’ desktops will need to have Java (at least 1.8) correctly installed.
In the startup script (.bat or .sh file) for the cu.jar file, launch the app via the following command:

```
java -jar cu.jar -DAUTH_ENDPOINT=<url of auth app in cloud run>
```

When giving the auth endpoint address, be sure to not have any trailing / symbols.

Other Configuration Options

The desktop app uses an internal producer/consumer model to generate file segments that are then uploaded and stitched. These configuration options allow tuning of the app to control the number of simultaneous uploads and the segments that can be queue in memory.

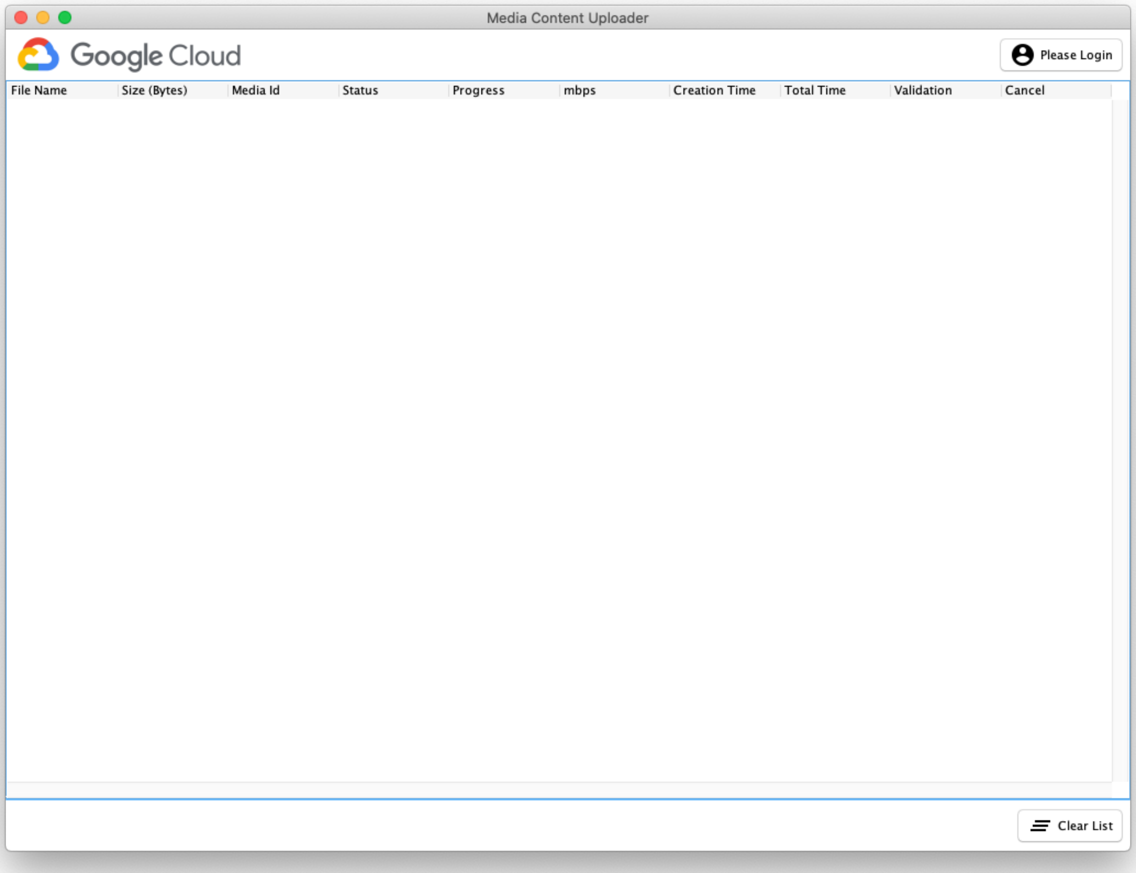
The desktop app can be tuned with various parameters:

```
java -jar cu.jar -DAUTH_ENDPOINT=<url of auth app in cloud run> \
     -DUPLOAD_QUEUE_SIZE=<upload_queue_size> \
     -DUPLOAD_WORKER_SIZE=<upload_worker_size> \
     -DTASK_WORKER_SIZE=<task_worker_size> \
     -DREQUIRE_MEDIA_ID=<true|false>
```

The config parameters work as follows:

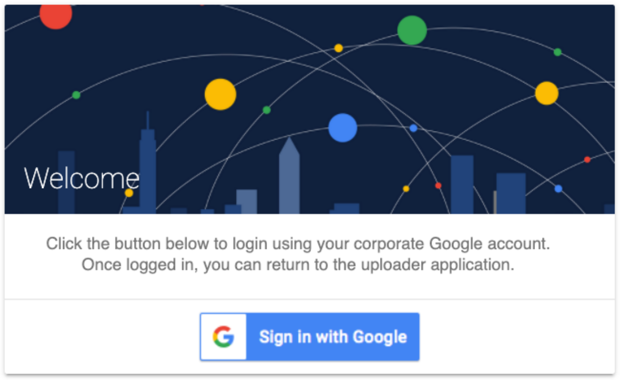
- **UPLOAD_QUEUE_SIZE**: (Default 20) this is the length of the number of items that will wait in the queue to be uploaded. For smaller memory machines, this can be reduced, since each segment is 48 MiBs. This is a blocking queue. When full (ie the actual uploads are taking too long), the producers will block on adding new segments for upload.
- **UPLOAD_WORKER_SIZE**: (Default 3) this is the number of parallel uploads. For machines with higher network bandwidth, this can be increased to get higher overall throughput
- **TASK_WORKER_SIZE**: (Default 10) this is the number of task workers that execute other tasks in the system (such as checksumming, stitching, file segmenting, etc.). This can be left as is in most cases. Since the upload is blocking, increasing the task worker pool doesn't really impact overall throughput on faster machines.
- **REQUIRE_MEDIA_ID**: (Default 'true') when true, will require that users input a media id before upload can commence. When 'false', a media id will be optional: if one is entered it will be accepted, but if it is absent, there will be no error thrown and the upload will proceed as normal.

Screen Views



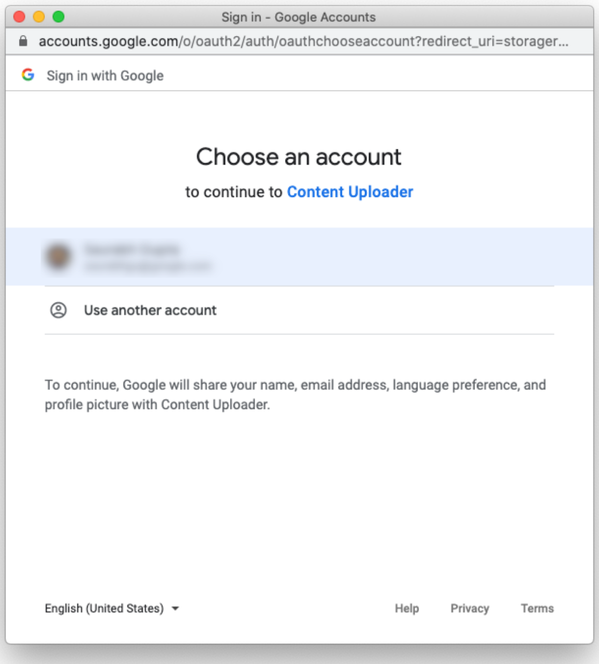
Above: App is launched and user presses the "Please Login" button

Media Content Uploader Login


Welcome

Click the button below to login using your corporate Google account.
Once logged in, you can return to the uploader application.

Above: User sees the web login form using Google Auth


Sign in - Google Accounts

accounts.google.com/o/oauth2/auth/oauthchooseaccount?redirect_uri=storager...

Sign in with Google

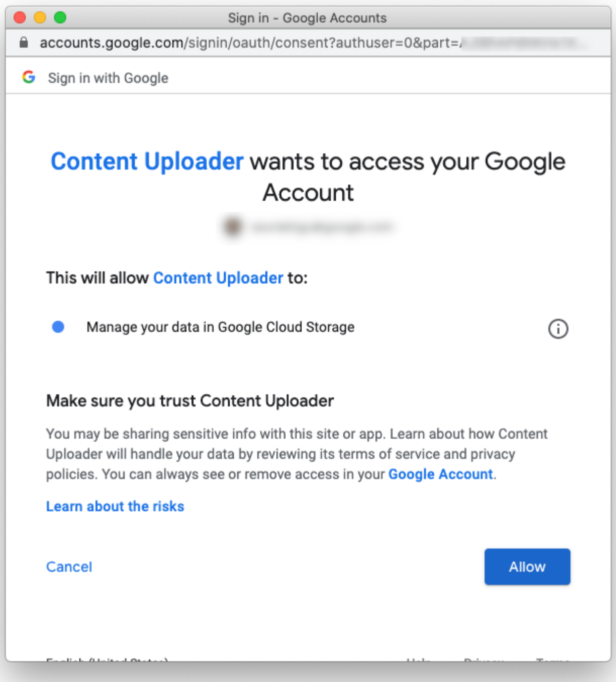
Choose an account

to continue to Content Uploader

Use another account

To continue, Google will share your name, email address, language preference, and profile picture with Content Uploader.

Above: user is asked for the correct account use for login



Sign in - Google Accounts

accounts.google.com/signin/oauth/consent?authuser=0&part=[redacted]

Sign in with Google

Content Uploader wants to access your Google Account

[redacted]

This will allow Content Uploader to:

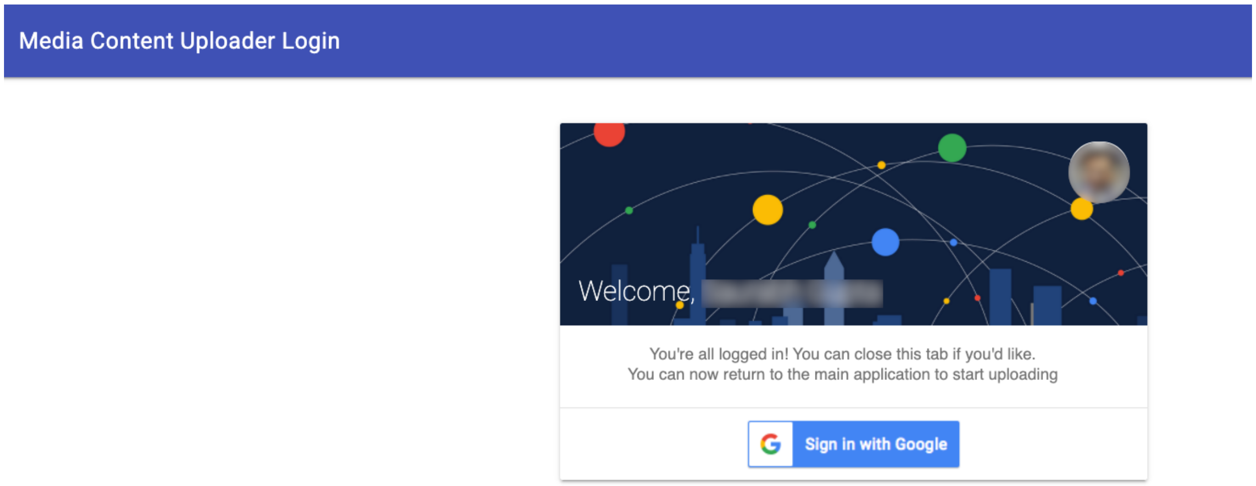
- Manage your data in Google Cloud Storage

Make sure you trust Content Uploader

You may be sharing sensitive info with this site or app. Learn about how Content Uploader will handle your data by reviewing its terms of service and privacy policies. You can always see or remove access in your Google Account.

Learn about the risks

Above: User selects the account and Google confirms the authorization requested by the app

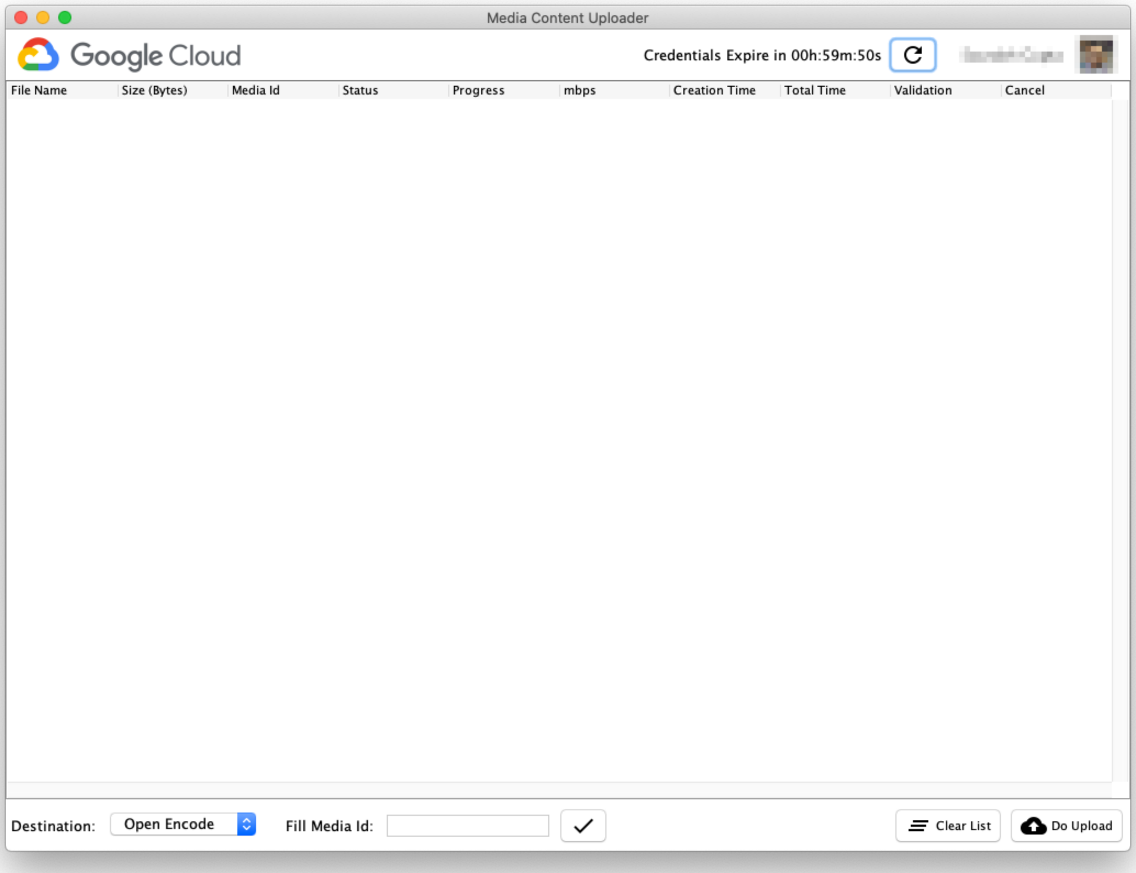


Media Content Uploader Login

Welcome, [redacted]

You're all logged in! You can close this tab if you'd like.
You can now return to the main application to start uploading

Above: User is returned to the web login with their login information showing on the page as confirmation



Media Content Uploader

Google Cloud

Credentials Expire in 00h:59m:50s

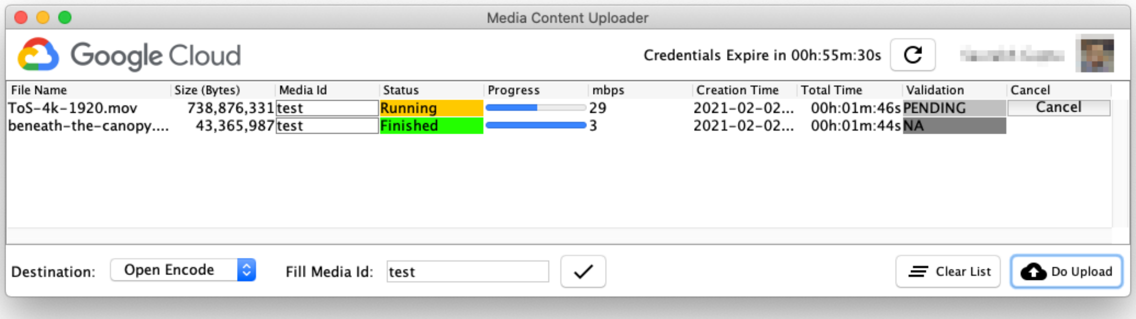
[redacted]

File Name	Size (Bytes)	Media Id	Status	Progress	mbps	Creation Time	Total Time	Validation	Cancel

Destination: Open Encode

Fill Media Id:

Above: user returns to the app and can see their name and avatar on the top right corner as confirmation of login; the credentials will auto renew after 1 hr



Media Content Uploader

Google Cloud

Credentials Expire in 00h:55m:30s

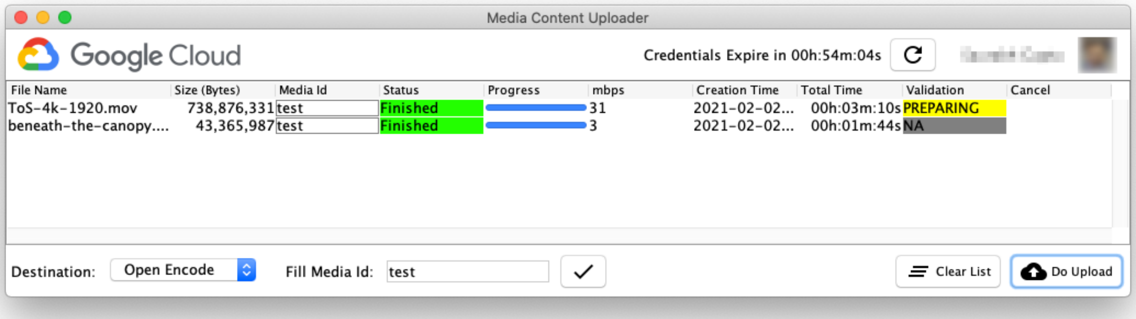
[redacted]

File Name	Size (Bytes)	Media Id	Status	Progress	mbps	Creation Time	Total Time	Validation	Cancel
ToS-4k-1920.mov	738,876,331	test	Running		29	2021-02-02...	00h:01m:46s	PENDING	Cancel
beneath-the-canopy....	43,365,987	test	Finished		3	2021-02-02...	00h:01m:44s	NA	

Destination: Open Encode

Fill Media Id: test

Above: user drags and drops some large files and clicks “Do Upload”; the uploads start with the progress showing as needed



Media Content Uploader

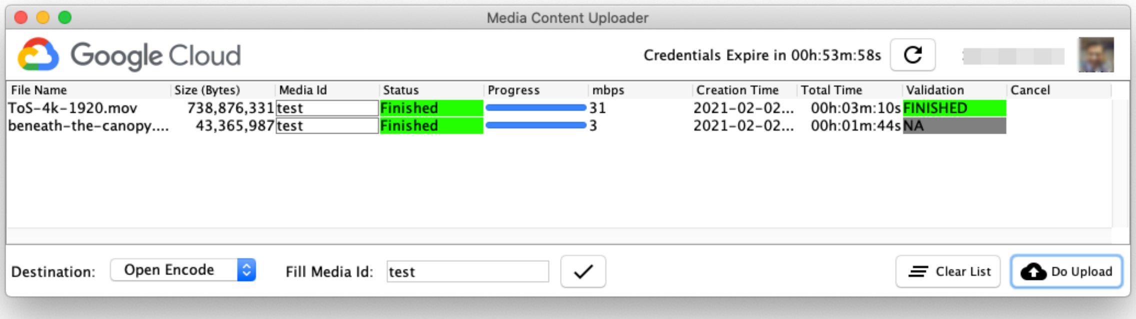
Google Cloud

Credentials Expire in 00h:54m:04s [redacted]

File Name	Size (Bytes)	Media Id	Status	Progress	mbps	Creation Time	Total Time	Validation	Cancel
ToS-4k-1920.mov	738,876,331	test	Finished		31	2021-02-02...	00h:03m:10s	PREPARING	
beneath-the-canopy....	43,365,987	test	Finished		3	2021-02-02...	00h:01m:44s	NA	

Destination: Open Encode   Fill Media Id: test

Above: the large file (>250mb) is preparing to be stitched (the "validation" column is showing "Preparing")



Media Content Uploader

Google Cloud

Credentials Expire in 00h:53m:58s [redacted]

File Name	Size (Bytes)	Media Id	Status	Progress	mbps	Creation Time	Total Time	Validation	Cancel
ToS-4k-1920.mov	738,876,331	test	Finished		31	2021-02-02...	00h:03m:10s	FINISHED	
beneath-the-canopy....	43,365,987	test	Finished		3	2021-02-02...	00h:01m:44s	NA	

Destination: Open Encode   Fill Media Id: test

Above: the large file has been successfully uploaded and validated after stitching the composite uploads and CRC32c checking