5 - Convert Videos Hosted on Google Cloud Storage

23:30

Convert Videos Hosted on Google Cloud Storage

Instead of just processing videos locally, we will now process videos that are hosted on Google Cloud Storage. The whole point of this service is that we will process videos after they are uploaded by users, so we need some external storage for that.

The general flow of the service will be:

1. A user uploads a video to Google Cloud Storage
2. The `video processing service` will be notified of the upload via Cloud Pub/Sub
3. The `video processing service` will download the video from Google Cloud Storage
4. The `video processing service` will process the video

5. The `video processing service` will upload the processed video to Google Cloud Storage

In this lesson, we will write code to cover the steps for 3, 4 and 5.

We will discuss steps 1 and 2 in much more detail in upcoming lessons.

Mark Lesson Complete

2. Create a `storage.ts` File

This file will contain all of the code that interacts with either Google Cloud Storage, or our local file system.

Copy

```
import { Storage } from "@google-cloud/storage";
import fs from 'fs';
import ffmpeg from 'fluent-ffmpeg';


const storage = new Storage();

const rawVideoBucketName = "neetcode-yt-raw-videos";
const processedVideoBucketName = "neetcode-yt-processed-videos";

const localRawVideoPath = "./raw-videos";
const localProcessedVideoPath = "./processed-videos";

/**
 * Creates the local directories for raw and processed videos.
 */
export function setupDirectories() {
  ensureDirectoryExistence(localRawVideoPath);
  ensureDirectoryExistence(localProcessedVideoPath);
}


/**
 * @param rawVideoName - The name of the file to convert from {@link
localRawVideoPath}.
 * @param processedVideoName - The name of the file to convert to
{@link localProcessedVideoPath}.
 * @returns A promise that resolves when the video has been converted.
 */
export function convertVideo(rawVideoName: string, processedVideoName:
string) {
  return new Promise<void>((resolve, reject) => {
    ffmpeg(`${localRawVideoPath}/${rawVideoName}`)
      .outputOptions("-vf", "scale=-1:360") // 360p
      .on("end", function () {
        console.log("Processing finished successfully");
        resolve();
      })
      .on("error", function (err: any) {
        console.log("An error occurred: " + err.message);
        reject(err);
      })
      .save(`${localProcessedVideoPath}/${processedVideoName}`);
  });
}


/**
 * @param fileName - The name of the file to download from the
 * {@link rawVideoBucketName} bucket into the {@link
localRawVideoPath} folder.
 * @returns A promise that resolves when the file has been downloaded.
 */
export async function downloadRawVideo(fileName: string) {
  await storage.bucket(rawVideoBucketName)
    .file(fileName)
    .download({
      destination: `${localRawVideoPath}/${fileName}`,
    });

  console.log(
    `gs://${rawVideoBucketName}/${fileName} downloaded to
${localRawVideoPath}/${fileName}.`
  );
}


/**
 * @param fileName - The name of the file to upload from the
 * {@link localProcessedVideoPath} folder into the {@link
processedVideoBucketName}.
 * @returns A promise that resolves when the file has been uploaded.
 */
export async function uploadProcessedVideo(fileName: string) {
  const bucket = storage.bucket(processedVideoBucketName);

  // Upload video to the bucket
  await storage.bucket(processedVideoBucketName)
    .upload(`${localProcessedVideoPath}/${fileName}`, {
      destination: fileName,
    });
  console.log(
    `${localProcessedVideoPath}/${fileName} uploaded to
gs://${processedVideoBucketName}/${fileName}.`
  );

  // Set the video to be publicly readable
  await bucket.file(fileName).makePublic();
}
```

```
/**
 * @param fileName - The name of the file to delete from the
 * {@link localRawVideoPath} folder.
 * @returns A promise that resolves when the file has been deleted.
 *
 */
export function deleteRawVideo(fileName: string) {
  return deleteFile(`${localRawVideoPath}/${fileName}`);
}


/**
* @param fileName - The name of the file to delete from the
* {@link localProcessedVideoPath} folder.
* @returns A promise that resolves when the file has been deleted.
*
*/
export function deleteProcessedVideo(fileName: string) {
  return deleteFile(`${localProcessedVideoPath}/${fileName}`);
}


/**
 * @param filePath - The path of the file to delete.
 * @returns A promise that resolves when the file has been deleted.
 */
function deleteFile(filePath: string): Promise<void> {
  return new Promise((resolve, reject) => {
    if (fs.existsSync(filePath)) {
      fs.unlink(filePath, (err) => {
        if (err) {
          console.error(`Failed to delete file at ${filePath}`, err);
          reject(err);
        } else {
          console.log(`File deleted at ${filePath}`);
          resolve();
        }
      });
    } else {
      console.log(`File not found at ${filePath}, skipping delete.`);
      resolve();
    }
  });
}


/**
 * Ensures a directory exists, creating it if necessary.
 * @param {string} dirPath - The directory path to check.
 */
function ensureDirectoryExistence(dirPath: string) {
  if (!fs.existsSync(dirPath)) {
    fs.mkdirSync(dirPath, { recursive: true }); // recursive: true
enables creating nested directories
    console.log(`Directory created at ${dirPath}`);
  }
}
```

3. Update `index.ts`

```
import express from 'express';

import {
  uploadProcessedVideo,
  downloadRawVideo,
  deleteRawVideo,
  deleteProcessedVideo,
  convertVideo,
  setupDirectories
} from './storage';

// Create the local directories for videos
setupDirectories();

const app = express();
app.use(express.json());

// Process a video file from Cloud Storage into 360p
app.post('/process-video', async (req, res) => {

  // Get the bucket and filename from the Cloud Pub/Sub message
  let data;
  try {
    const message = Buffer.from(req.body.message.data,
'base64').toString('utf8');
    data = JSON.parse(message);
    if (!data.name) {
      throw new Error('Invalid message payload received.');
    }
  } catch (error) {
    console.error(error);
    return res.status(400).send('Bad Request: missing filename.');
  }

  const inputFileName = data.name;
  const outputFileName = `processed-${inputFileName}`;

  // Download the raw video from Cloud Storage
  await downloadRawVideo(inputFileName);

  // Process the video into 360p
  try {
    await convertVideo(inputFileName, outputFileName)
  } catch (err) {
    await Promise.all([
      deleteRawVideo(inputFileName),
      deleteProcessedVideo(outputFileName)
    ]);
    return res.status(500).send('Processing failed');
  }

  // Upload the processed video to Cloud Storage
  await uploadProcessedVideo(outputFileName);

  await Promise.all([
    deleteRawVideo(inputFileName),
    deleteProcessedVideo(outputFileName)
  ]);

  return res.status(200).send('Processing finished successfully');
});

const port = process.env.PORT || 3000;
app.listen(port, () => {
    console.log(`Server is running on port ${port}`);
});
```

References

1. Cloud Storage Client Library: **https://cloud.google.com/storage/docs/reference/libraries#client-libraries-usage-nodejs**
2. Using Pub/Sub with Cloud Run tutorial: **https://cloud.google.com/run/docs/tutorials/pubsub#run_pubsub_server-nodejs**