



Images haven't loaded yet. Please exit printing, wait for images to load, and try to print again.

Mar 2, 2017 · 5 min read

## File Storage with Google Cloud Storage

*On 2/28/2017 Amazon AWS S3 had an outage which affected a lot of websites. This got me thinking if Google Cloud offered a solution which was similar to S3. In this blog I will be creating a simple API where you can upload a file which would upload it to Google Cloud Storage.*

*I will be using restify to on the backend and using boilerplate I created, which can be found [here](#).*

. . .

First clone the boilerplate locally using the following command

```
git clone git@github.com:binoy14/restify-boilerplate.git
GCloudStorage
```

Once inside the folder, we will install google cloud storage's npm module, which can be found [here](#).

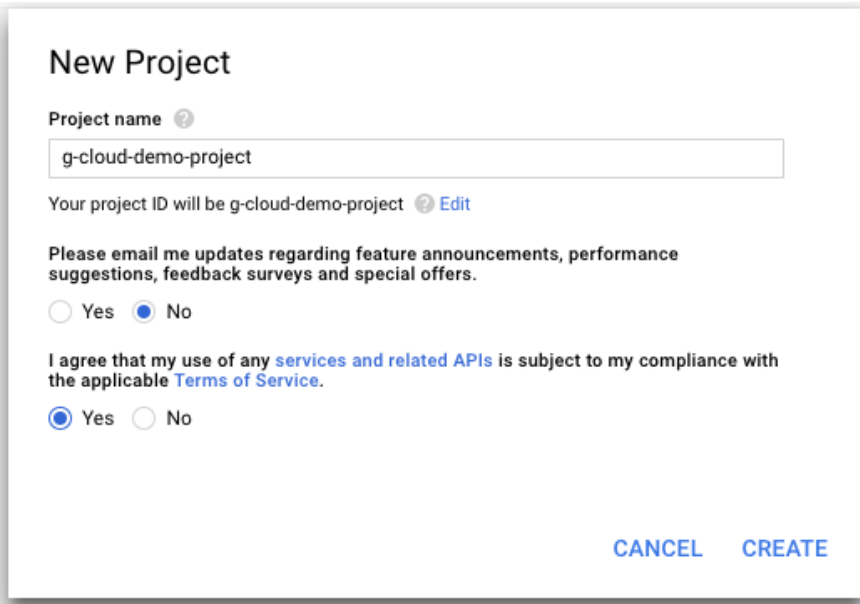
```
yarn add @google-cloud/storage

or

npm install --save @google-cloud/storage
```

Before we write any code, we need to configure Google Cloud Storage on [Google Developer Console](#). Follow the following steps to get the Cloud Platform ready for use.

1. Create a new project on [Google Developer Console](#)

A screenshot of the 'New Project' dialog box in the Google Cloud console. It features a title 'New Project', a 'Project name' field with a help icon and the text 'g-cloud-demo-project', and a confirmation of the project ID. Below this, there are two sections of opt-in emails, each with 'Yes' and 'No' radio buttons. The first section is for feature announcements, and the second is for terms of service. At the bottom right are 'CANCEL' and 'CREATE' buttons.

**New Project**

Project name [?](#)

g-cloud-demo-project

Your project ID will be g-cloud-demo-project [?](#) [Edit](#)

Please email me updates regarding feature announcements, performance suggestions, feedback surveys and special offers.

☐ Yes ☒ No

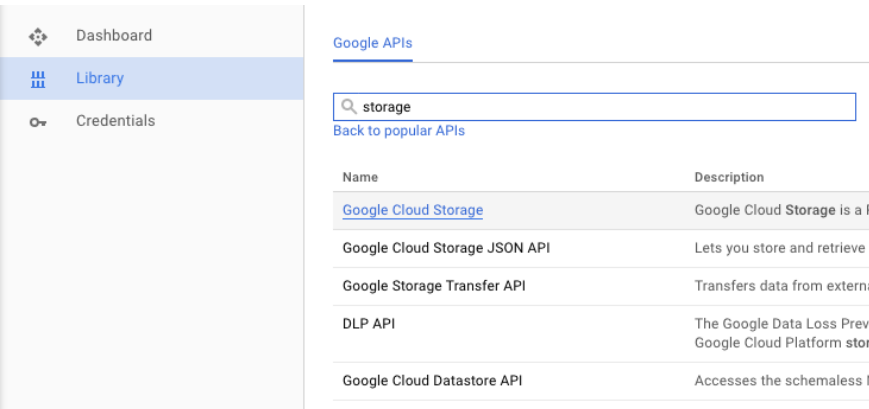
I agree that my use of any [services and related APIs](#) is subject to my compliance with the applicable [Terms of Service](#).

☒ Yes ☐ No

[CANCEL](#) [CREATE](#)

Create New Project Dialogue

2. Enable Google Cloud Storage for your project. This API can be found by clicking on Library in the sidebar and in the searching for storage

A screenshot of the Google APIs library page. The left sidebar shows 'Dashboard', 'Library' (selected), and 'Credentials'. The main area is titled 'Google APIs' and contains a search bar with 'storage' entered. Below the search bar is a link 'Back to popular APIs'. A table lists search results with columns 'Name' and 'Description'. The first result, 'Google Cloud Storage', is highlighted.

Dashboard

Library

Credentials

Google APIs


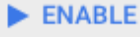
storage


[Back to popular APIs](#)

Name	Description
<a href="#">Google Cloud Storage</a>	Google Cloud Storage is a R
Google Cloud Storage JSON API	Lets you store and retrieve f
Google Storage Transfer API	Transfers data from externa
DLP API	The Google Data Loss Prev Google Cloud Platform stor
Google Cloud Datastore API	Accesses the schemaless h

Searching for Google Cloud Storage API

Click on Enable in the Google Cloud Storage Page

 **Google Cloud Storage** 

 This API is part of Google Cloud Platform. [Go to this product in the Cloud](#)

### About this API

Google Cloud Storage is a RESTful service for storing and accessing your d

#### Using credentials with this API

##### Accessing user data with OAuth 2.0

You can access user data with this API. On the Credentials page, create an client ID. A client ID requests user consent so that your app can access use Include that client ID when making your API call to Google. [Learn more](#)

#### Server-to-server interaction

You can use this API to perform server-to-server interaction, for example be web application and a Google service. You'll need a service account, which app-level authentication. You'll also need a service account key, which is us authorize your API call to Google. [Learn more](#)

Enable Google Cloud Storage API

3. To use this API we will need to create credentials and to do so we will navigate to credentials page by clicking on **Credentials** in the sidebar. On the create credentials dropdown choose **service account key** option. And when on create service account page we will choose the below config and download the JSON file inside our project.

## Credentials

Credentials OAuth consent screen Domain verification

API key  
Identifies your project using a simple API key to check quota and access

OAuth client ID  
Requests user consent so your app can access the user's data

Service account key  
Enables server-to-server, app-level authentication using robot accounts

Help me choose  
Asks a few questions to help you decide which type of credential to use

Create credentials

Credentials Page

## Create service account key

## Service account

Compute Engine default service account

## Key type

Downloads a file that contains the private key. Store the file securely because this key can be recovered if lost.

☒ JSON

Recommended

☐ P12

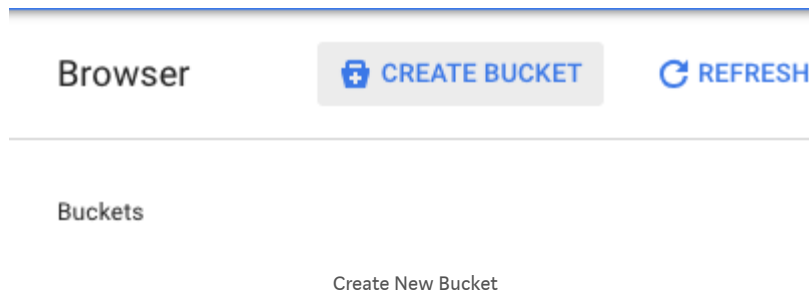
For backward compatibility with code using the P12 format

Create

Cancel

Create Service Account Key Configuration

4. Next, we need to create a bucket on Google Cloud Storage. *Note: The name of the bucket needs to be unique*



5. (Optional) The Final step is to make our files read public so we can return a URL from our API. To do so we need to install Google Cloud SDK using this [link](#) and run the following command.

```
gsutil defacl ch -u allUsers:R gs://your-bucket-name-here
```

. . .

Once all the configuration is done we are ready to write code to upload our file to Google Cloud Storage.

First thing first is we are going to add a new route for upload in *src/routes/index.js* Add the following

```
server.post("/user/upload", User.uploadUser);
```

Next thing we need to do is export the uploadUser function in *src/routes/users/index.js*

```
import getUsers from "./getUsers";
import getUser from "./getUser";
import postUser from "./postUser";
import putUser from "./putUser";
import deleteUser from "./deleteUser";
import uploadUser from "./uploadUser";

module.exports = {
  getUser,
  getUsers,
  postUser,
```

```
putUser,  
deleteUser,  
uploadUser,  
};
```

and finally create a new file called *uploadUser.js* inside *src/routes/users/*

```
1  const fs = require("fs");  
2  const restify = require("restify");  
3  const uuidv4 = require("uuid/v4");  
4  const Storage = require("@google-cloud/storage");  
5  const CLOUD_BUCKET = "your-bucket-name-here";  
6  const storage = Storage({  
7    projectId: 'your-project-id',  
8    keyFilename: '/path/to/file/downloaded/in/step/3'  
9  })  
10 const bucket = storage.bucket(CLOUD_BUCKET);  
11  
12 const uploadUser = (req, res) => {  
13   const {file} = req.files;  
14   const gcsname = uuidv4() + file.name;  
15   const files = bucket.file(gcsname);  
16  
17   fs.createReadStream(file.path)  
18     .pipe(files.createWriteStream({  
19       metadata: {  
20         contentType: file.type  
21       }  
22     })))
```

TLDR; This file is the controller for */user/upload* route. This controller receives the file in the request and using streams the file is uploaded to Google Cloud Storage and the public url of the file is returned in the response.

```
const Storage = require("@google-cloud/storage");  
const CLOUD_BUCKET = "your-bucket-name-here";  
const storage = Storage({  
  projectId: 'your-project-id',  
  keyFilename: '/path/to/file/downloaded/in/step/3'
```

```
});  
const bucket = storage.bucket(CLOUD_BUCKET);
```

We will be importing Storage module which we installed previously. We will set the bucket name, projectId and in the keyFileName we will put the path of the file downloaded in step 3 and finally we will create a bucket constant which will be reference to our bucket.

```
const {file} = req.files;  
const gcsname = uuidv4() + file.name;  
const files = bucket.file(gcsname);
```

The file object is in located in *req.files*. We will need to create a unique file name for the file which needs to be uploaded. To do so we will use node's uuid module to create a unique hash and append the file's name in the end. Finally we will add the file to our bucket using *bucket.file(gcsname)*

```
fs.createReadStream(file.path)  
  .pipe( files.createWriteStream({  
    metadata: {  
      contentType: file.type  
    }  
  }))  
  .on("error", (err) => {  
    res.status(500).send(restify.InternalServerError(err));  
  })  
  .on('finish', () => {  
    res.json({  
      success: true,  
      fileUrl:  
        `https://storage.googleapis.com/${CLOUD_BUCKET}/${gcsname}`  
    })  
  });
```

We will use node's *fs* module and pipe *createWriteStream* which is part of the *file* function from *google-cloud/storage*. We also set metadata of *contentType* to information from our file. We will listen to error and finish events from *createReadStream*. In the error event we will simply return **restify.InternalServerError(err)**, which just return a status code of 500. In the finish event we return the *fileUrl* as the public url of

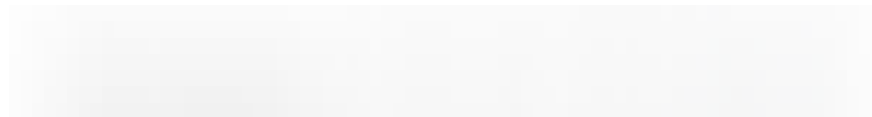
the file. which will be at

```
https://storage.googleapis.com/CLOUD_BUCKET_NAME/FILE_NAME
```

To test this we can create a simple form in HTML using the following code.

```
<html>
<head>
<title>File Upload</title>
</head>
<body>
<form action="http://localhost:3030/user/upload"
method="POST" enctype="multipart/form-data">
  <input type="file" name="file">
  <button>Submit</button>
</form>
</body>
</html>
```

Once the file is uploaded we can go to Google Cloud Storage website and see that our file is there and if we open the link we will be able to download our file as well.



File on Google Cloud Storage

. . .

This was a simple tutorial on uploading file to Google Cloud Storage. There is a lot of other features provided by the API which are not demonstrated in this blog. Checkout [API Documentation](#) for more functions that are available.

. . .

*If you liked this, click the ❤️ below and share. Follow for more content.*





