Project 1

Jaygna Mehta

jmehta2024@fau.edu

GCP Project Name: Jaygna Upload Image

Project ID: jaygnauploadimage

GCP Project Console: <https://console.cloud.google.com/home/dashboard?invt=AboL0g&project=jaygnauploadimage>

App URL: <https://fauuploadimage-659617434688.us-central1.run.app>

GitHub Repository Link: <https://github.com/jaygna/fauuploadimage>

# Introduction

The Project is about to upload Image and store to the data storage in Goggle cloud in the bucket list.

# Architecture

#### 1. **Google Cloud Run (App Deployment and Execution)**

* **Role**: Cloud Run will host the Flask application in a serverless environment. It allows you to run the app in the cloud without managing servers. Cloud Run scales automatically based on incoming requests.

#### 2. **Google Cloud Storage (Image Storage)**

* **Role**: Cloud Storage is where the uploaded images will be stored in a bucket. The Flask app will interact with this service to upload, retrieve, and display images.

#### 3. **Flask Web Application (Backend & Logic)**

* **Role**: The Flask app serves as the core backend for processing user requests. It handles the image upload logic, storage interaction, displaying uploaded images, and serving the frontend HTML.

#### 4. **Frontend (HTML Interface)**

* **Role**: The frontend HTML page will provide a user interface for uploading and visualizing images. It will send the images to the Flask backend, which will then upload them to Cloud Storage

# Implementation

 **Create a Google Cloud Storage Bucket**:

* Navigate to Google Cloud Console.
* Go to **Storage** -> **Browser**.
* Click **Create bucket** and follow the instructions to create a new storage bucket where your images will be stored.

 **Enable necessary APIs**:

* Enable the **Cloud Run API**.
* Enable the **Cloud Storage API**.
* Enable the **Cloud IAM API**.

 **Create Service Account for Cloud Run**:

* Navigate to **IAM & Admin** -> **Service Accounts**.
* Create a new service account and grant it roles:

 **Download Service Account Credentials**:

* After creating the service account, create and download the JSON key file. You'll need this to authenticate your application to Google Cloud.

# Pros and Cons

#### 1. **Scalability and Auto-scaling (Cloud Run)**

#### 2. **Serverless Architecture**

#### 3. **Cost Efficiency**

#### 4. **Easy Integration with Google Cloud Services**

#### 5. **Simplified Deployment Process**

#### 1. **Complexity in Handling Large-Scale Image Storage**

#### 2. **Limited Customization for Serverless Platforms**

#### 3. **File Size and Upload Limits**

#### 4. **Security Concerns (Public Access)**

# Problems encountered and Solutions

Problem: File Upload Size Limits

Solution – May be we can modify the code to upload larger files.

# Lessons learned

* + 1. How to create a datastore and give access to users
    2. How to integrate the storage code to bucket in datastore

# Appendix

1 main.py

import os

from flask import Flask, redirect, request, send\_file

from google.cloud import datastore, storage

import time

# Initialize Google Cloud clients

datastore\_client = datastore.Client()

storage\_client = storage.Client()

# Initialize Flask app

app = Flask(\_\_name\_\_)

# Create 'files' directory if it doesn't exist

os.makedirs('files', exist\_ok=True)

# Google Cloud Storage bucket name

BUCKET\_NAME = 'fau-image-storage'

# Routes

@app.route('/')

def index():

    index\_html = """

    <form method="post" enctype="multipart/form-data" action="/upload">

      <div>

        <label for="file">Choose file to upload</label>

        <input type="file" id="file" name="form\_file" accept="image/jpeg"/>

      </div>

      <div>

        <button>Submit</button>

      </div>

    </form>

    """

    for file in list\_files():

        index\_html += "<li><a href='/files/" + file + "'>" + file + "</a></li>"

    return index\_html

@app.route('/upload', methods=["POST"])

def upload():

    file = request.files['form\_file']  # The file item name must match the form's input name

    if file:

        # Save to Google Cloud Storage

        file\_name = file.filename

        save\_to\_gcs(file\_name, file)

        # Add metadata to Google Cloud Datastore

        metadata = {

            "name": file\_name,

            "url": f'https://storage.googleapis.com/{BUCKET\_NAME}/{file\_name}',

            "user": "rdeandrade",  # You can customize this based on the actual user

            'timestamp': int(time.time())

        }

        add\_db\_entry(metadata)

    return redirect("/")

@app.route('/files')

def list\_files():

    # List files in Google Cloud Storage bucket

    return get\_list\_of\_files(BUCKET\_NAME)

@app.route('/files/<filename>')

def get\_file(filename):

    # Serve file from Google Cloud Storage

    return send\_file\_from\_gcs(BUCKET\_NAME, filename)

### Google Cloud Storage functions ###

def get\_list\_of\_files(bucket\_name):

    """Lists all the blobs in the bucket."""

    print(f"Fetching file list from bucket: {bucket\_name}")

    blobs = storage\_client.list\_blobs(bucket\_name)

    return [blob.name for blob in blobs]

def save\_to\_gcs(file\_name, file):

    """Upload file to Google Cloud Storage."""

    print(f"Uploading {file\_name} to {BUCKET\_NAME}")

    bucket = storage\_client.bucket(bucket\_name=BUCKET\_NAME)

    blob = bucket.blob(file\_name)

    blob.upload\_from\_file(file)

    print(f"File {file\_name} uploaded successfully.")

def send\_file\_from\_gcs(bucket\_name, file\_name):

    """Serve a file from Google Cloud Storage."""

    blob = storage\_client.bucket(bucket\_name).blob(file\_name)

    return blob.public\_url

### Google Cloud Datastore functions ###

def add\_db\_entry(metadata):

    """Add metadata to Google Cloud Datastore."""

    entity = datastore.Entity(key=datastore\_client.key('photos'))

    entity.update(metadata)

    datastore\_client.put(entity)

def fetch\_db\_entry(query\_filters):

    """Fetch entries from Datastore based on query filters."""

    query = datastore\_client.query(kind='photos')

    for attr, value in query\_filters.items():

        query.add\_filter(attr, '=', value)

    return list(query.fetch())

### Main execution ###

if \_\_name\_\_ == '\_\_main\_\_':

    app.run(debug=True, port=5002)

2 Procfile

web: gunicorn --bind :$PORT --workers 1 --threads 8 --timeout 0 main:app

3 requirements.txt

# https://pypi.org/project/flask

Flask==3.0.2

# https://pypi.org/project/gunicorn

gunicorn==21.2.0