Project 1

Ibrahim Luke Barhoumeh
ibarhoumeh2022@fau.edu
GCP Project Name: ibarhoumeh-cot5930P1
GCP Project ID: ibarhoumeh-cot5930p1
GCP Project Console: PROJECT LINK
Cloud Run Service name: imaging-app
App URL: https://imaging-app-856239458079.us-east1.run.app/
GitHub Repository Link: https://github.com/lukebarhoumeh/COT5930-project1

Introduction

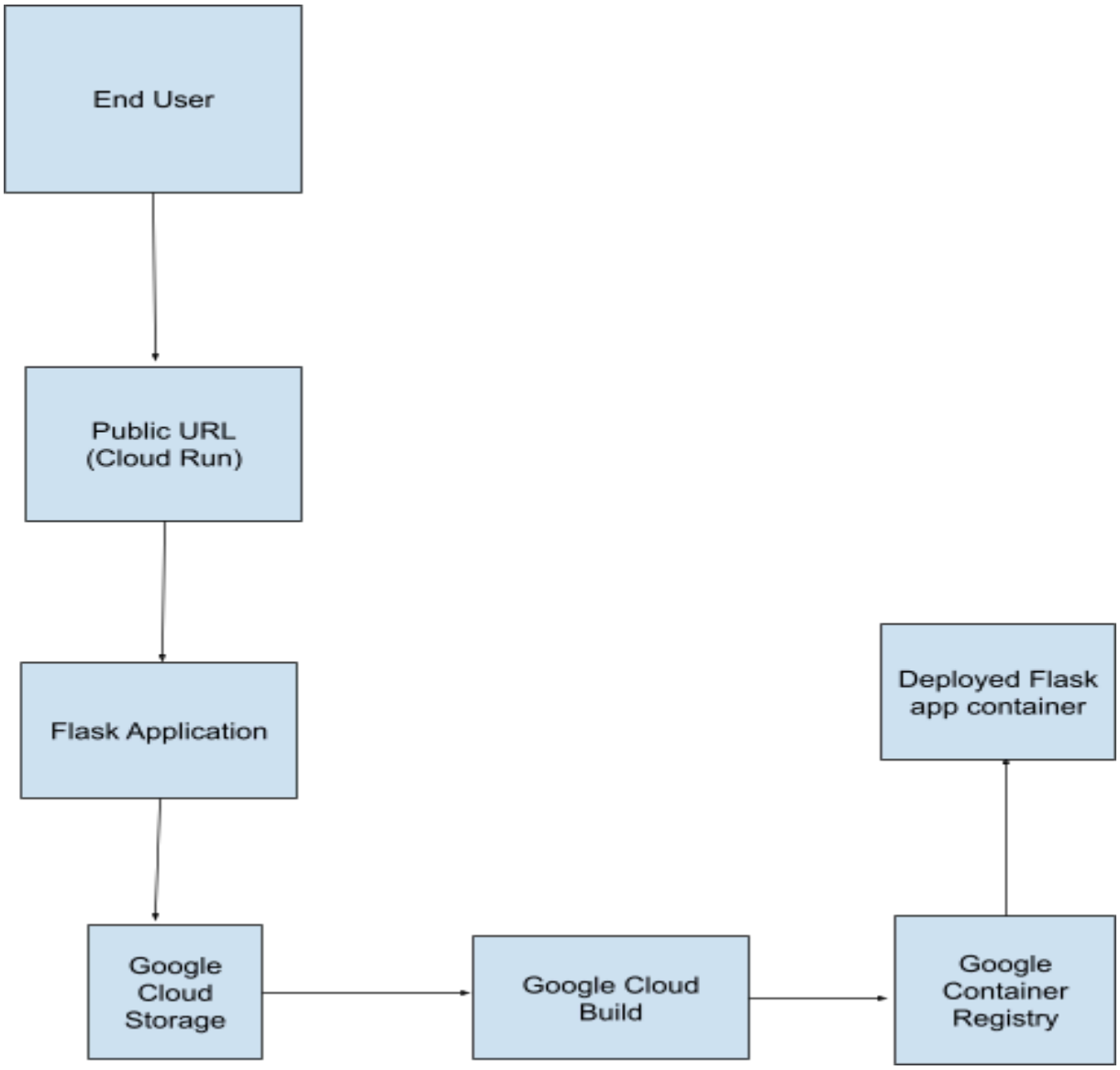
This project focuses on creating a cloud-native app that allows user to upload and visualize images using python and flask. The goal of this project was to implement a scalable, cloud deployed app that uses what we learned in class as well as using our resources. The objective of this project was creating a web interface for users to upload image files, storing those files in a google storage bucket, and providing an accessible platform for retrieving and viewing those uploaded images.

Implemented features

- Image upload: users can upload .jpeg or .jpg images through HTML interface
- Cloud storage integration: automatically stored in a google cloud storage bucket for secure and scalable storage
- File visualization: a list of uploaded files that is presented dynamically, allowing users to view and download the stored images
- Cloud run: the app is deployed via cloud run, ensuring its scalability and availability, also making it public access URL.

Architecture

The architecture for this project was set to leverage all of google cloud services and modern cloud native principals to ensure scalability and user accessibility. Below is an overview of the components and their role in the overall architecture



Components involved:

- End user
- Cloud run(hosting Flask App)
- Google Cloud Storgage (bucket)
- Container registry
- Google Cloud Build

1. End users
    a. Interaction: End users access the app via web using public cloud URL
    b. Purpose Upload images and view a list of existing images
2. Public URL (Cloud Run)
    a. Role: Interface that bridges end users with the app
    b. Purpose: secure and publicly accessible URL to the deployed Flask App
    c. Implementation: automatically generated by Google Cloud Run when app is deployed
3. Cloud Run(Flask app)
    a. Role: core app hosting flask app
    b. Purpose: handles user requests like uploading files and fetching existing images
    c. Features
        i. Receives uploaded images via POST requests
        ii. Retrieves a list of stored images from Google Cloud Storage
        iii. Displays uploaded images
    d. Containerization: Built user a Dockerfile. Deployed as a container image
4. Google Cloud Storage (Bucket)
    a. Role: persistent storage for uploaded images
    b. Bucket Structure
        i. Primary bucket: cot5930-image-p1, stores images in a dedicated folder called files/
    c. Features:
        i. Allows uploading and retrieving images via the Flask app
        ii. Organizses images for scalility and accessibility
5. Google Cloud Build
    a. Role: Continuous integration for building and deploying the app
    b. Purpose: Automatically complies the Dockerfile into a container image and pushes it to Google Container registry
6. Google Container Registry
    a. Roles: Stores the built container image for deployment
    b. Purpose: Provides version-controlled Storage of the container image for use in Cloud run

Implementation

Code Structure:

- The main.py file contains the core logic of the Flask app
    - Endpoints
        - / (GET): displays HTML interface for users to upload images and view uploaded files
        - /upload (POST): Handles image uploads via HTTP POST requests
        - /files (GET): lists all files stored in the Google Cloud Storage Bucket
        - /files/<filename> (GET): retrieves and displays an individual file from the bucket

    - Dependencies:
        - Flask: For creating and managing the web application
        - Google-cloud-storage: For interacting with the Google Cloud Storage bucket
        - Os: managing file paths locally
        - requests : For handling HTTP requests
    - The Storage.py file contains the helper functions to interact with Google Cloud Storage:
        - Upload_file: Uploads images to the bucket
        - Get_list_of_files: lists all stored images in the bucket
        - Download_file: Downloads specific files from the bucket when requested
    - The requirements.txt file has all necessary python dependencies to make sure app runs correctly

- Features:
    - Accepts only JPEG images for upload
    - Displays a dynamically generated list of uploaded images via HTML
    - Handles errors

- Configuration:
    - Port Number: Configured to run on port 8080
    - Startup Command:
        - Locally: main.py
        - Cloud Run: Dockerfile sets up the environment and serves the app using flask

- HTML interface:
    - Located in the / endpoint
    - Contains an image upload form <form>, with fields for selecting and a “Submit” button
    - Dynamically generates links to view uploaded files via flasks response to /files

Cloud Run

- Purpose: Hosts Flask app in a managed, serverless environment
- Configuration:
    - Cloud Run was set up using the gcloud
    - The Dockerfile ensures the app is properly containerized
        - Specifies a python base image
        - Installs required dependencies
        - Sets the working directory, copies project files, and defines the app entry point (main.py)

- Public Access
    - Configured Cloud run to allow public access to the deployed app using IAM settings

Google Cloud Storage (Bucket)

- Bucket Details:
    - Bucket Name: cot5930-image-p1
    - Structure: Images are stored in the files/ folder within the bucket for organization
- Configuration:
    - Created through the Google Cloud Console
    - Permissions: Configured for public file access to allow reriveal of images via the Flask app
    - Used google-cloud-storage python library for interaction:
        - Uploading images: upload_file
        - Listing images: get_list_of_files
        - Retrieving images: download_files

Google Cloud Build

- Purpose: Automated containerization and deployment of the app
- Setup:
    - Configured gcloud commands(gcloud builds submit --tag gcr.io/ibarhoumeh-cot5930p1/imaging-app)
    - Links github repo to the build process for integration

Google Container Registry

- Purpose: Stores the built Docker image for the Flask app
- Setup:
    - The containerized image pushed to the registry as gcr.io/ibarhoumeh-cot5930p1/image-app

Github repo

- Details
    - Has all .py files, Dockerfile, requirements.txt, and other project files
    - Tracks all code changes with commits
    - Public Repo

Pros and Cons

PROS:

- Serverless Scalability
    - Using GC run allows the app to scale automatically, based on user demand, ensuring cost efficiency and resource optimization
- Simplicity and Accessibility
    - The solution uses Flask, lightweight and easy to develop and maintain
    - The URL generated by Cloud Run provides a straightforward way for users to access the app
- Centralized Storage
    - Images stored in the Google Cloud Storage, providing reliable, secure, and globally available storage. Ensuring the data is easily accessible and can handle large volumes of data with minimal configuration
- Automated Deployment:
    - Google Cloud Build and container Registry streamline deployment process by automating containerization and linking the Github repo. Improving deployment workflow and ensures the app is always up to date
- Modularity:
    - Separation of logic into main.py(Flask App) and storage.py(Cloud Storage interaction) ensures code maintainability and makes future enhancements easier.
- Error Handling and File Management:
    - The app is set up to handle errors, such as unsupported file types, and provides clear feedback to users
    - Organizing uploaded files in a dedicated folder within the bucket to improve managebility

CONS

- Limited Performance for High Traffic:
    - Flask, may struggle under heavy loads without additional support like multi threading or deploying multiple instances via cloud run
- No Authentication:
    - The app lacks user authentication and authorization, meaning it wont differentiate or restrict access based on user roles. Leading to potential misuse if implemented for production
- Bucket Permissions:
    - Public access to the bucket, while convenient for project, intros security risks, implement pre signed URLs
- Basic UX:
    - HTML interface is funcation, lacks features such as drag and drop uploads, real time updates, or image previews
- Single Storage Bucket:
    - Using a single bucket for all uploads could become a problem as the number of files increases.

Problems encountered and Solutions

- Missing dependencies in the environment
    - While running Flask locally, ModuleNotFoundError occurred for Google CLoud Storage module
    - Cause: Required dependencies weerent installed in python
    - Solution:
        - Added all necessary dependencies to a requirements.txt file
        - Installed dependencies using pip install -r requirements.txt in the virtual environment

- Port conflict during testing
    - Probelm: running app locally resulted in an error stating that defualt flask port(5000) was in use
    - Cause: Another app or previous Flask instance was running on the same port
    - Solution:
        - Used 1sof -i:5000 to identify the process using the port
        - Terminated the process with kill -9 <PID>
        - Updated the Flask app to use port 8080

- Errors in Dockerfile Configuration
    - Problem: Deployment failed during Cloud Build due to an incomplete Dockerfile configuration.
    - Cause: The Dockerfile did not include the correct setup for dependencies and file structure.
    - Solution:
        - Modified the Dockerfile to:
        - Use an Python base image.
        - Install dependencies from requirements.txt.
        - Copy necessary project files and set the working directory.
        - Specify the Flask app entry point (main.py).

- Public Access Denied to Cloud Run
    - Problem: The deployed Cloud Run app returned a "403 Forbidden" error when accessed via the public URL.
    - Cause: IAM permissions for the Cloud Run service were not configured for public access.
    - Solution:

        - Updated IAM settings to include the allUsers role with Cloud Run Invoker permissions.
        - Confirmed public accessibility through the app's URL.

- Bucket Permission Issues
    - Problem: Uploaded images in the Google Cloud Storage bucket were inaccessible due to insufficient permissions.
    - Cause: The bucket's permissions were not set up for public access to files.
    - Solution:
        - Configured the bucket to allow public read access for the files/ folder.
        - Verified that images were retrievable through the Flask app using the appropriate endpoints.

- Difficulty Synchronizing GitHub Repository
    - Problem: Conflicts occurred when pushing changes from Cloud Shell to the GitHub repository.
    - Cause: Code in Cloud Shell was out of sync with the GitHub repository.
    - Solution:
        - Resolved conflicts using git pull --rebase.
        - Pushed the finalized code using git push -f after ensuring no data was lost.

Application instructions

- Access the Application
    - Open a web browser and navigate to the app's public URL: https://imaging-app-856239458079.us-east1.run.app.

- Upload an Image
    - Click the "Choose File" button to select a JPEG image from your device.
    - Once selected, click the "Submit" button to upload the image.

- View Uploaded Images
    - After uploading, you will see a dynamically generated list of uploaded images on the same page.
    - Click on any image name from the list to view it directly in your browser.

- Retrieve Files
    - Use the /files/<filename> endpoint by appending the file name to the app URL to retrieve and download a specific image.

- Error Handling
    - If a non-JPEG file is uploaded, the application will return an error message indicating invalid file format.

Lessons learned

- Learned how to design and implement a cloud-native application by using Google Cloud services like Cloud Run, Google Cloud Storage, and Google Cloud Build
- Learned how to structure Flask App with multiple Endpoints to handle different functionality
- Learned how to containerize apps using Docker
- Integration between Google CloudBuild with container registry
-

Appendix

ALL IN GITHUB REPO