Project: Deploying a Portfolio Website on Google Cloud Run with CI/CD

Project Overview

In this project, you'll be developing and deploying a website on Google Cloud Run. The project involves setting up a **multi-stage CI/CD pipeline** to build, test, and deploy your code automatically. You'll use **Docker** to package your application, **Cloud Build** to handle building and pushing images, and **GitHub Actions** to set up continuous integration and delivery. By the end, you'll have a fully automated system that deploys your changes to the cloud every time you push an update to GitHub.

What You'll Need

- **Google Cloud Platform (GCP)** account: You can start with Google's free trial credits.
- **GitHub** account: This is where you'll store your code and set up your CI/CD pipeline.
- **Basic familiarity with Docker** and the terminal.

Step 1: Set Up Your Project Repository on GitHub

1. **Create a New Repository on GitHub**:
    - Go to GitHub, log in, and create a new repository. Name it something like `portfolio-website` or `flask-app-portfolio`.
2. **Clone the Repository**:
    - Clone the repository to your local machine so you can start adding files and organizing the project.

    - Clone the repository of any static portfolio website you want to use and move the files and folders in it into your own cloned repository on your local.

```
git clone https://github.com/your-username/portfolio-website.git
cd portfolio-website
```

3. **Organize Project Files**:
    - **README.md**: Add a README file to describe your project.
    - **Folder Structure**:
        - For a **static site** (e.g., HTML, CSS, JS), place these files directly in the repository.

For a **Flask app**, create a folder structure like this:

```
├── app/
│   ├── main.py  # Your main Flask file
│   ├── templates/
│   └── static/
└── Dockerfile
```

- Add a `.gitignore` file to exclude unnecessary files, like `__pycache__`, `*.env`, or `node_modules/` (if applicable).

Step 2: Build Your Application

1. **Develop Your Application**:
   - **Static Site**: Add your HTML, CSS, and JavaScript files.

Flask App: If you're building with Flask, create a simple app in `main.py`:
python

```python
# app/main.py
from flask import Flask
app = Flask(__name__)

@app.route("/")
def home():
    return "<h1>Welcome to my portfolio!</h1>"

if __name__ == "__main__":
    app.run(host="0.0.0.0", port=8080)
```

Step 3: Write the Dockerfile

A **Dockerfile** is needed to package your application so it can run consistently anywhere.

1. **For a Static Site Using NGINX**:
   - We'll use `nginx:alpine` (a lightweight web server) to serve your static files.
   - Create a nginx.conf file in the root directory and paste this code into it.

nginx.conf

```
server {
    listen 8080;
    server_name localhost;
```

```
    location / {
        root /usr/share/nginx/html;
        index index.html;
    }
}
```

This is to ensure that nginx is serving from port 8080 and not 80 which can lead to this deployment error

```
Deployment failed
ERROR: (gcloud.run.deploy) Revision 'portfolio-site-00002-dx6' is not ready and cannot serve traffic. The user-provided container failed to start and listen on the port defined provided by the PORT=8080 environment variable within the allocated timeout. This can happen when the container port is misconfigured or if the timeout is too short. The healtcheck timeout can be extended. Logs for this revision might contain more information.
```

-
- Create a file called Dockerfile in the root directory and paste this code into it.

Dockerfile (The D must be capital)

```
# Start from the official Nginx image  and use a lightweight web server to serve static files
FROM nginx:alpine

#Copy the content of the nginx config file into this nginx root file
COPY nginx.conf /etc/nginx/conf.d/default.conf

# Copy your website files into the Nginx web directory
COPY . /usr/share/nginx/html

# Expose port 80 to access your website
EXPOSE 8080

# Start Nginx server
CMD ["nginx", "-g", "daemon off;"]
```

2. **For a Flask Application**:
    - We'll use `python:3.9-slim` to run the Flask application.

Dockerfile

```
Copy code
# Use a lightweight Python image
```

```
FROM python:3.9-slim
WORKDIR /app
COPY . /app

# Install dependencies
RUN pip install -r requirements.txt

# Start the Flask app
CMD ["python", "app/main.py"]
```

3. **Testing Locally**:
    - For windows OS, ensure your docker desktop app is opened and running or go to the start menu to open the app before building your docker image.
    - Build and run your Docker image locally to make sure everything works before deploying.

bash

```
docker build -t portfolio-site .
docker run -p 8080:80 portfolio-site  # For NGINX
docker run -p 8080:8080 portfolio-site  # For Flask
```

Step 4: Set Up Google Cloud

1. **Sign up to a free trial on google cloud**
2. **Create a Google Cloud Project**:
    - Go to Google Cloud Console and create a new project.
3. **Enable APIs**:
    - Download the google cloud CLI/SDK on your local and follow the prompt to install it.
    - Go to your local terminal (power shell or git bash) to initialise and confirm that gcloud cli is running correctly

```
gcloud init # To initialise the google cloud Cli
gcloud – version
```

    - You'll need to enable **Cloud Run** and **Cloud Build** for deploying and building Docker images by typing the following commands into your terminal

bash

```
gcloud services enable run.googleapis.com
```

```
gcloud services enable cloudbuild.googleapis.com
```

4. **Set Up Authentication**:
    - Authenticate your local machine with Google Cloud by running:

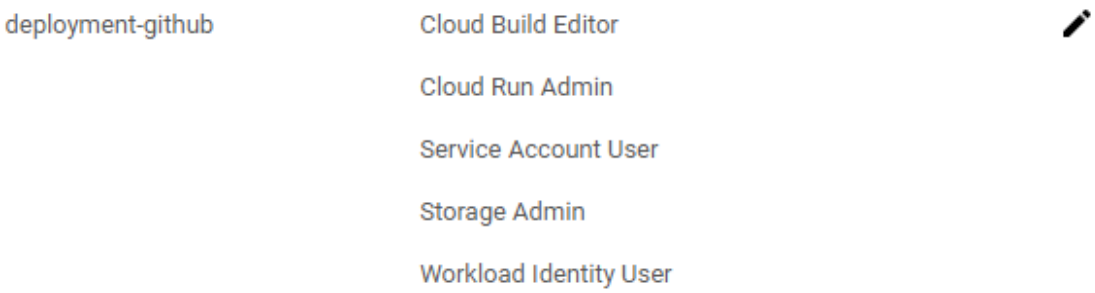
bash

```
gcloud auth login
```

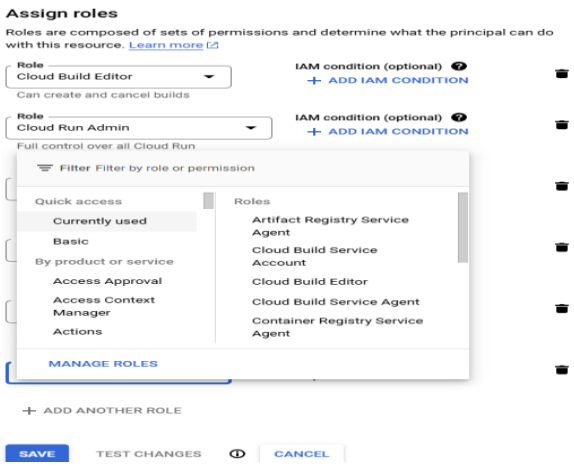
5. **IAM Permissions**:

   Set up permissions for your Google Cloud project to allow deployment to Cloud Run.

    - **Create a Service Account**:
        i. In the Google Cloud Console, navigate to **IAM & Admin > Service Accounts**.
        ii. Click **Create Service Account** and follow the steps to create an account with deployment permissions.
            1. Ensure your service account has the following roles by going to the service plan created and clicking on the pencil icon beside it to add the following roles.



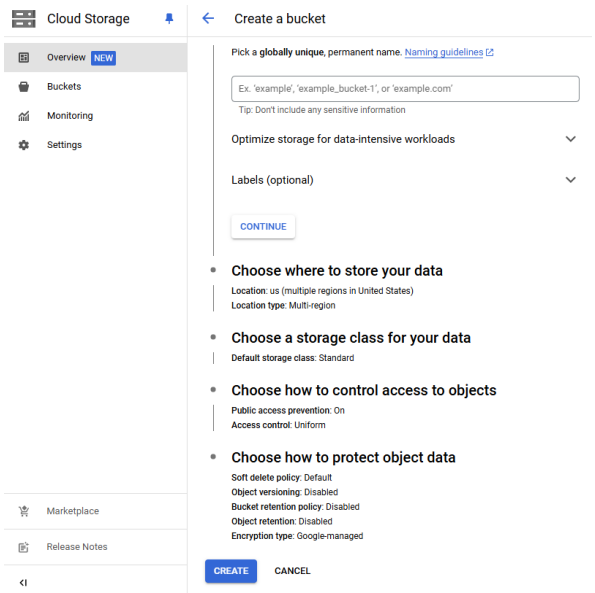
            2. Click add another role and type the role name in the search box



iii. **Download the Service Account Key**:
   1. After creating the service account, go to **Keys** and **Add Key > Create new key**.
   2. Choose JSON format, and download the file.

iv. **Add the JSON Key to GitHub:**
   1. **In your GitHub repository, go to Settings > Secrets and Variables > Actions.**
   2. **Click New repository secret and add**
      a. `Name:` `GCP_CREDENTIALS`
      Secret: JSON credentials downloaded from your Google Cloud service account.
      Click the New repository secret again
      b. `Name:` `GCP_PROJECT`
      Secret: Your Google Cloud project ID.

6. **CREATE A BUCKET IN YOUR GCP PROJECT**
   - **Go to cloud storage > buckets > +Create**
   - Type your bucket name, leave the default settings and click create
     Note: The bucket name will be used in your deploy.yml file to keep the deployment logs

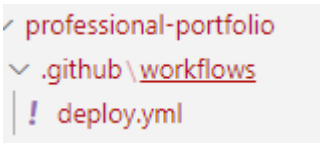


Step 5: Set Up CI/CD with GitHub Actions and Cloud Build

1. **Why We Need Cloud Build and GitHub Actions**:
   - **GitHub Actions**: This will handle the CI/CD pipeline. Every time you push code to GitHub, GitHub Actions will trigger a workflow to deploy the code.

- **Cloud Build**: Cloud Build is Google's tool for building and pushing Docker images to their Container Registry. It's the recommended way to deploy Docker images to Cloud Run.

2. **GitHub Actions Workflow Configuration**:
   - In your repository, create a folder structure in the root folder like `.github/workflows` and add a file called `deploy.yml` to have a folder structure like this

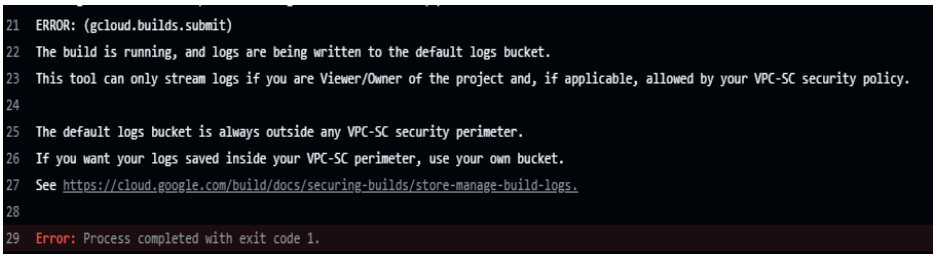


yaml

```yaml
name: CI/CD for Portfolio Site

on:
  push:
    branches:
      - main

jobs:
  build-and-deploy:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout code
        uses: actions/checkout@v2

      - name: Authenticate with GCP
        uses: google-github-actions/auth@v2
        with:
          credentials_json: ${{ secrets.GCP_CREDENTIALS }}

      - name: Trigger Cloud Build
        run: |
          gcloud builds submit --tag gcr.io/${{ secrets.GCP_PROJECT
}}/portfolio-site:${{ github.sha }} --gcs-log-dir gs://<change to your
bucket name without this <> symbol>


      - name: Deploy to Cloud Run
        run: |
          gcloud run deploy portfolio-site \
            --image gcr.io/${{ secrets.GCP_PROJECT }}/portfolio-site:${{
github.sha }} \
```

```
            --platform managed \
            --region us-central1 \
            --allow-unauthenticated
```

3. **Explanation of Steps**:
    - **Authenticate with GCP**: We authenticate GitHub Actions to access Google Cloud.
    - **Trigger Cloud Build**: This command tells Google Cloud Build to take your code, build it into a Docker image, and push it to Google's Container Registry then keep the logs in your bucket and not the default log bucket to avoid this error:



    - **Deploy to Cloud Run**: Finally, we deploy the Docker image to Cloud Run.
4. **GitHub Secrets**:
    - In your repository's settings, add secrets:
        - `GCP_CREDENTIALS`: JSON credentials from your Google Cloud service account.
        - `GCP_PROJECT`: Your Google Cloud project ID.

Step 6: Testing and Troubleshooting

1. **Push Your Code**:
    - Make any changes, commit, and push to the `main` branch.

bash

```
git add .
git commit -m "Deploying portfolio site"
git push origin main
```

2. **View Workflow in GitHub Actions**:
    - Go to the Actions tab in your GitHub repository to monitor the progress. You can check logs for each step to see if there are any issues.
3. **Check Cloud Run Deployment**:
    - Go to Cloud Run in Google Cloud Console to see your deployed service.

- Copy the service URL and open it in your browser to view your deployed website.

Step 7: Document the Project

1. **Create Documentation with GitHub Pages**:
   - Use GitHub Pages to write documentation about your project. You can explain:
     - How you built the Dockerfile.
     - Steps in your GitHub Actions workflow.
     - Any challenges or mistakes you encountered.

Potential Mistakes to Avoid

1. **Missing Permissions**: If you get a permission error, double-check your IAM permissions on GCP.
2. **Invalid Secrets**: Ensure you're correctly setting up `GCP_CREDENTIALS` and `GCP_PROJECT` in GitHub.
3. **Build Failures**: Check the logs in GitHub Actions to understand where errors occur, whether during build or deployment.
