Flask App Engine Deployment Guide

A comprehensive guide to deploying Flask applications on Google App Engine Flexible, with face detection using Cloud Vision API.

Table of Contents

Overview

What is App Engine?

Google App Engine is a fully managed platform that automatically handles:

- ***Auto-scaling based on traffic***
- ***Load balancing across instances***
- ***Version management and rollbacks***
- ***Security scanning and monitoring***
- ***Native integration with Google Cloud services***

Project Features

This sample application demonstrates:

- ***Flask web framework for Python***
- ***Cloud Vision API for face detection***
- ***Cloud Storage for image hosting***
- ***Datastore (NoSQL database) for metadata storage***
- ***App Engine Flexible for deployment***

Lab Setup

1. Get Sample Code

```
# Download the sample application
gcloud storage cp -r gs://spls/gsp023/flex_and_vision/ .
cd flex_and_vision
```

2. Authentication Setup

```
# Set project ID environment variable
export PROJECT_ID=$(gcloud config get-value project)

# Create service account
gcloud iam service-accounts create qwiklab \
  --display-name "My Qwiklab Service Account"

# Grant owner role to service account
gcloud projects add-iam-policy-binding ${PROJECT_ID} \
  --member serviceAccount:qwiklab@${PROJECT_ID}.iam.gserviceaccount.com \
  --role roles/owner

# Create service account key
gcloud iam service-accounts keys create ~/key.json \
  --iam-account qwiklab@${PROJECT_ID}.iam.gserviceaccount.com

# Set credentials environment variable
export GOOGLE_APPLICATION_CREDENTIALS="/home/${USER}/key.json"
```

3. Create App Engine Instance

```
# Set your region
AE_REGION=us-east4

# Create App Engine app
gcloud app create --region=$AE_REGION
```

4. Create Cloud Storage Bucket

```
# Set bucket name (using project ID)
export CLOUD_STORAGE_BUCKET=${PROJECT_ID}

# Create the bucket
gsutil mb gs://${PROJECT_ID}
```

Local Development

Setting Up Virtual Environment

```
# Create Python 3 virtual environment
virtualenv -p python3 env

# Activate virtual environment
source env/bin/activate

# Install dependencies
pip install -r requirements.txt
```

Running Locally

```
# Start the Flask application
python main.py
```

Visit `http://localhost:8080` in your browser to test the application.

To stop: Press `CTRL+C`

Application Architecture

Project Structure

```
flex_and_vision/
├── templates/
│   └── homepage.html     # Jinja2 HTML template
├── app.yaml              # App Engine configuration
├── main.py               # Flask application
└── requirements.txt      # Python dependencies
```

Key Code Components

1. Importing Google Cloud Libraries

```
from google.cloud import datastore
from google.cloud import storage
from google.cloud import vision
```

2. Homepage Route

Fetches stored images from Datastore and renders the template:

```
@app.route('/')
def homepage():
    # Create Datastore client
    datastore_client = datastore.Client()

    # Query for all face entities
    query = datastore_client.query(kind='Faces')
    image_entities = list(query.fetch())

    # Render template with image data
    return render_template('homepage.html', image_entities=image_entities)
```

3. Saving to Datastore

Stores image metadata after face detection:

```
# Create Datastore client
datastore_client = datastore.Client()

# Get current timestamp
current_datetime = datetime.now()

# Define entity kind and name
kind = 'Faces'
name = blob.name

# Create entity key
key = datastore_client.key(kind, name)

# Construct entity with properties
entity = datastore.Entity(key)
entity['blob_name'] = blob.name
entity['image_public_url'] = blob.public_url
entity['timestamp'] = current_datetime
entity['joy'] = face_joy

# Save to Datastore
datastore_client.put(entity)
```

4. HTML Template (homepage.html)

Uses Jinja2 templating to display images:

```
<form action="upload_photo" method="POST" enctype="multipart/form-data">
  Upload File: <input type="file" name="file"><br>
  <input type="submit" name="submit" value="Submit">
</form>

{% for image_entity in image_entities %}
  <img src="{{image_entity['image_public_url']}}" width=200 height=200>
  <p>{{image_entity['blob_name']}} was uploaded {{image_entity['timestamp']}}.</p>
  <p>Joy Likelihood for Face: {{image_entity['joy']}}</p>
{% endfor %}
```

Deployment

1. Configure app.yaml

Edit the configuration file:

```
nano app.yaml
```

Update with your settings:

```
runtime: python
env: flex
entrypoint: gunicorn -b :$PORT main:app

runtime_config:
    operating_system: "ubuntu22"
    runtime_version: "3.12"

env_variables:
  CLOUD_STORAGE_BUCKET: <your-project-id>

manual_scaling:
  instances: 1
```

Important: Replace `<your-project-id>` with your actual project ID.

Save and exit (CTRL+X, then Y, then ENTER).

2. Set Cloud Build Timeout

gcloud config set app/cloud_build_timeout 1000

3. Deploy to App Engine

gcloud app deploy

Type Y when prompted. Deployment takes approximately 10 minutes.

4. Access Your Application

 # Your app will be available at:

https://<PROJECT_ID>.appspot.com

 # Or open it directly:

gcloud app browse

Applying This to Your Projects

Running This Locally (Outside Lab Environment)

Prerequisites

Install Google Cloud SDK

 # macOS (using Homebrew)

brew install --cask google-cloud-sdk

 # Linux

curl https://sdk.cloud.google.com | bash

exec -l $SHELL

 # Windows: Download installer from cloud.google.com/sdk

1.

Install Python 3.8+

python3 --version

2.
3. *Create a Google Cloud Project*

    - *Visit console.cloud.google.com*
    - *Create new project or select existing one*
    - *Enable billing (required for App Engine and APIs)*

Local Setup Steps

```
# 1. Authenticate with Google Cloud
gcloud auth login
gcloud auth application-default login

# 2. Set your project
gcloud config set project YOUR_PROJECT_ID

# 3. Enable required APIs
gcloud services enable vision.googleapis.com
gcloud services enable datastore.googleapis.com
gcloud services enable storage.googleapis.com
gcloud services enable appengine.googleapis.com

# 4. Create your project directory
mkdir my-flask-vision-app
cd my-flask-vision-app

# 5. Set up virtual environment
python3 -m venv env
source env/bin/activate  # On Windows: env\Scripts\activate

# 6. Create requirements.txt
cat > requirements.txt << EOF
Flask==2.3.0
gunicorn==21.2.0
google-cloud-vision==3.4.0
google-cloud-storage==2.10.0
google-cloud-datastore==2.15.0
EOF

# 7. Install dependencies
pip install -r requirements.txt

# 8. Create service account and download key
gcloud iam service-accounts create my-app-sa --display-name "My App Service Account"
gcloud projects add-iam-policy-binding YOUR_PROJECT_ID \
```

```
  --member="serviceAccount:my-app-sa@YOUR_PROJECT_ID.iam.gserviceaccount.com" \
  --role="roles/owner"
gcloud iam service-accounts keys create ./key.json \
  --iam-account=my-app-sa@YOUR_PROJECT_ID.iam.gserviceaccount.com

# 9. Set environment variables
export GOOGLE_APPLICATION_CREDENTIALS="$(pwd)/key.json"
export CLOUD_STORAGE_BUCKET="YOUR_PROJECT_ID"
```

Adapting for Your Own Projects

1. Basic Flask + Google Cloud Template

```
# minimal_app.py
from flask import Flask, render_template, request
from google.cloud import storage
from google.cloud import vision
from google.cloud import datastore
import os

app = Flask(__name__)

# Initialize Google Cloud clients
storage_client = storage.Client()
vision_client = vision.ImageAnnotatorClient()
datastore_client = datastore.Client()

BUCKET_NAME = os.environ.get('CLOUD_STORAGE_BUCKET')

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/upload', methods=['POST'])
def upload():
    file = request.files['file']
    # Add your upload and processing logic here
    return "File uploaded successfully!"

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

2. Common Use Cases

Image Analysis Service

- *Use Vision API for: label detection, text extraction, landmark detection*
- *Store results in Datastore for quick retrieval*
- *Host processed images in Cloud Storage*

Document Processing Pipeline

- *Accept file uploads (PDF, images)*
- *Use Vision API for OCR*
- *Store extracted text in Datastore*
- *Generate searchable document database*

Content Moderation System

- *Check uploaded images for inappropriate content*
- *Use Vision API's Safe Search detection*
- *Auto-flag problematic content*

3. Essential app.yaml Template

```
runtime: python
env: flex
entrypoint: gunicorn -b :$PORT main:app

runtime_config:
  operating_system: "ubuntu22"
  runtime_version: "3.12"

env_variables:
 CLOUD_STORAGE_BUCKET: YOUR_BUCKET_NAME
 # Add other environment variables here

automatic_scaling:
 min_num_instances: 1
 max_num_instances: 5
 cool_down_period_sec: 120
 cpu_utilization:
  target_utilization: 0.6
```

4. Cost Optimization Tips

- *Use Standard Environment (not Flex) for simpler apps to reduce costs*
- *Implement caching to reduce API calls*
- *Set scaling limits to prevent unexpected bills*
- *Delete unused resources (old versions, storage buckets)*

- *Monitor usage in Cloud Console billing section*

5. Security Best Practices

```
# Never commit credentials
# Add to .gitignore:
# key.json
# *.json
# .env

# Use environment variables for sensitive data
import os
SECRET_KEY = os.environ.get('SECRET_KEY')

# Validate file uploads
ALLOWED_EXTENSIONS = {'png', 'jpg', 'jpeg', 'gif'}

def allowed_file(filename):
    return '.' in filename and \
           filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS
```

Next Steps for Your Projects

1. *Start Simple: Build a basic Flask app locally first*
2. *Add One Cloud Service: Begin with just Storage or Datastore*
3. *Test Locally: Use emulators when possible (Datastore emulator)*
4. *Deploy Gradually: Test with manual scaling before enabling auto-scaling*
5. *Monitor Costs: Set up billing alerts in Google Cloud Console*
6. *Version Control: Use Git and keep credentials out of your repository*

Useful Resources

- *Flask Documentation*
- *Google Cloud Client Libraries*
- *App Engine Pricing Calculator*
- *Vision API Pricing*
- *Datastore Documentation*

Troubleshooting

Virtual environment not activating:

```
# On Windows, use:
```

env\Scripts\activate

 # On Unix/macOS:

source env/bin/activate

Authentication errors:

 # Re-authenticate

gcloud auth application-default login

Deployment timeout:

 # Increase timeout

gcloud config set app/cloud_build_timeout 2000

Port already in use:

 # Change port in main.py

app.run(host='0.0.0.0', port=8081)

Summary

This guide covered:

- ✅ *Deploying Flask apps to App Engine Flexible*
- ✅ *Using Google Cloud Vision, Storage, and Datastore*
- ✅ *Testing applications locally*
- ✅ *Setting up local development environment*
- ✅ *Adapting concepts for your own projects*

Key Takeaway: App Engine simplifies deployment and scaling, while Google Cloud services provide powerful capabilities without managing infrastructure.