

# Deploying a Python Flask Web Application to App Engine Flexible | Google Cloud Skills Boost

Qwiklabs : 17-21 minutes

---

## GSP023



## Google Cloud Self-Paced Labs

### Overview

In this lab you will learn how to deploy a Python Flask web application to the App Engine Flexible environment. The example application allows a user to upload a photo of a person's face and learn how likely it is that the person is happy. The application uses Google Cloud APIs for Vision, Storage, and Datastore.

### About App Engine

Google App Engine applications are easy to create, easy to maintain, and easy to scale as your traffic and data storage needs change. With App Engine, there are no servers to maintain. You simply upload your application and it's ready to go.

App Engine applications automatically scale based on incoming traffic. Load balancing, microservices, authorization, SQL and NoSQL databases, traffic splitting, logging, search, versioning, roll out and roll backs, and security scanning are all supported natively and are highly customizable.

App Engine's [Flexible Environment](#) supports a host of programming languages, including Java, Python, PHP, NodeJS, Ruby, and Go. App Engine's [Standard Environment](#) is an additional option for certain languages including Python. The two environments give users maximum flexibility in how their application behaves since each environment has certain strengths. Read [Choosing an App Engine Environment](#) for more information.

### What you'll learn

- How to deploy a simple web application to the App Engine Flexible Environment
- How to access the Google Cloud client libraries for Vision, Storage, and Datastore
- How to use Cloud Shell

### Prerequisites

- Familiarity with Python
- Familiarity with standard Linux text editors such as vim, emacs, or nano
- Access to an image with a face

# Setup and requirements

## Before you click the Start Lab button

Read these instructions. Labs are timed and you cannot pause them. The timer, which starts when you click **Start Lab**, shows how long Google Cloud resources will be made available to you.

This hands-on lab lets you do the lab activities yourself in a real cloud environment, not in a simulation or demo environment. It does so by giving you new, temporary credentials that you use to sign in and access Google Cloud for the duration of the lab.

To complete this lab, you need:

- Access to a standard internet browser (Chrome browser recommended).

**Note:** Use an Incognito or private browser window to run this lab. This prevents any conflicts between your personal account and the Student account, which may cause extra charges incurred to your personal account.

- Time to complete the lab---remember, once you start, you cannot pause a lab.

**Note:** If you already have your own personal Google Cloud account or project, do not use it for this lab to avoid extra charges to your account.

## How to start your lab and sign in to the Google Cloud Console

1. Click the **Start Lab** button. If you need to pay for the lab, a pop-up opens for you to select your payment method. On the left is the **Lab Details** panel with the following:
  - The **Open Google Console** button
  - Time remaining
  - The temporary credentials that you must use for this lab
  - Other information, if needed, to step through this lab

2. Click **Open Google Console**. The lab spins up resources, and then opens another tab that shows the **Sign in** page.

**Tip:** Arrange the tabs in separate windows, side-by-side.

**Note:** If you see the **Choose an account** dialog, click **Use Another Account**.

3. If necessary, copy the **Username** from the **Lab Details** panel and paste it into the **Sign in** dialog. Click **Next**.
4. Copy the **Password** from the **Lab Details** panel and paste it into the **Welcome** dialog. Click **Next**.

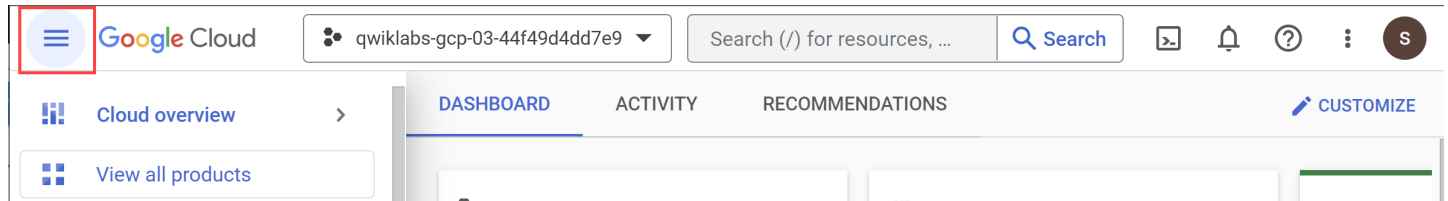
**Important:** You must use the credentials from the left panel. Do not use your Google Cloud Skills Boost credentials. **Note:** Using your own Google Cloud account for this lab may incur extra charges.

5. Click through the subsequent pages:
  - Accept the terms and conditions.

- Do not add recovery options or two-factor authentication (because this is a temporary account).
- Do not sign up for free trials.


After a few moments, the Cloud Console opens in this tab.

**Note:** You can view the menu with a list of Google Cloud Products and Services by clicking the **Navigation menu** at the top-left.



## Activate Cloud Shell

Cloud Shell is a virtual machine that is loaded with development tools. It offers a persistent 5GB home directory and runs on the Google Cloud. Cloud Shell provides command-line access to your Google Cloud resources.

1. Click **Activate Cloud Shell**  at the top of the Google Cloud console.

When you are connected, you are already authenticated, and the project is set to your **PROJECT\_ID**. The output contains a line that declares the **PROJECT\_ID** for this session:

Your Cloud Platform project in this session is set to YOUR\_PROJECT\_ID

gcloud is the command-line tool for Google Cloud. It comes pre-installed on Cloud Shell and supports tab-completion.

2. (Optional) You can list the active account name with this command:

```
gcloud auth list
```

3. Click **Authorize**.

4. Your output should now look like this:

### Output:

```
ACTIVE: * ACCOUNT: student-01-xxxxxxxxxxxx@qwiklabs.net To set the active account, run: $ gcloud
config set account `ACCOUNT`
```

5. (Optional) You can list the project ID with this command:

```
gcloud config list project
```

### Output:

```
[core] project = <project_ID>
```

### Example output:

[core] project = qwiklabs-gcp-44776a13dea667a6 **Note:** For full documentation of `gcloud`, in Google Cloud, refer to [the gcloud CLI overview guide](#).

## Task 1. Get the sample code

1. In Cloud Shell, run the following command to clone the Github repository:

```
gcloud storage cp -r gs://spls/gsp023/flex_and_vision/ .
```

2. Change directory into `flex_and_vision`:

```
cd flex_and_vision
```

## Task 2. Authenticate API requests

The Datastore, Storage, and Vision APIs are automatically enabled for you in this lab. In order to make requests to the APIs, you will need service account credentials. You can generate credentials from your project using `gcloud` in Cloud Shell. Your **Project ID** can be found on the tab where you started the lab.

1. Set an environment variable for your Project ID:

```
export PROJECT_ID=$(gcloud config get-value project)
```

2. Create a Service Account to access the Google Cloud APIs when testing locally:

```
gcloud iam service-accounts create qwiklab \ --display-name "My Qwiklab Service Account"
```

3. Give your newly created Service Account appropriate permissions:

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

4. After creating your Service Account, create a Service Account key:

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

This command generates a service account key stored in a JSON file named `key.json` in your home directory.

5. Using the absolute path of the generated key, set an environment variable for your service account key:

```
export GOOGLE_APPLICATION_CREDENTIALS="/home/${USER}/key.json"
```

Learn more about authenticating the Vision API from [Quickstart: Setup the Vision API Guide](#).

Click **Check my progress** below to check your lab progress.

Authenticate API Requests

## Task 3. Testing the application locally

## Starting your virtual environment and installing dependencies

1. Create an isolated Python 3 environment named `env` with [virtualenv](#):

```
virtualenv -p python3 env
```

2. Enter your newly created *virtualenv* named `env`:

```
source env/bin/activate
```

3. Use `pip` to install dependencies for your project from the `requirements.txt` file:

```
pip install -r requirements.txt
```

The `requirements.txt` file is a list of package dependencies you need for your project. The above command downloaded all of these listed package dependencies to the *virtualenv*.

## Creating an App Engine app

1. Next, create an App Engine instance by using:

```
gcloud app create
```

A prompt will display a list of regions.

2. Select a region that supports App Engine Flexible for Python then press ENTER.

You can learn more about regions and zones from the [Geography and regions Guide](#).

## Creating a storage bucket

1. First, set the environment variable `CLOUD_STORAGE_BUCKET` equal to the name of your `PROJECT_ID`. (It is generally recommended to name your bucket the same as your `PROJECT_ID` for convenience purposes):

```
export CLOUD_STORAGE_BUCKET=${PROJECT_ID}
```

2. Now run the following command to create a bucket with the same name as your `PROJECT_ID`:

```
gsutil mb gs://${PROJECT_ID}
```

Click **Check my progress** below to check your lab progress.

Create an App Engine App and Storage Bucket

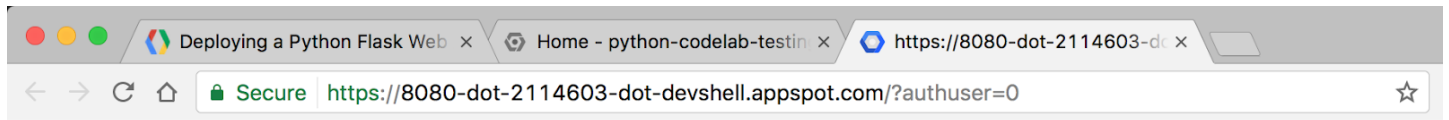
## Running the Application

1. Execute the following command to start your application:

```
python main.py
```

2. Once the application starts, click on the Web Preview icon  in the Cloud Shell toolbar and choose **Preview on port 8080**.

A tab in your browser opens and connects to the server you just started. You should see something like this:



## Google Cloud Platform - Face Detection Sample

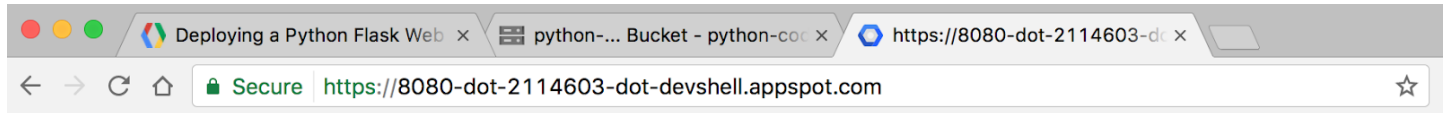
This Python Flask application demonstrates App Engine Flexible, Google Cloud Storage, Datastore, and the Cloud Vision API.

Upload File:  No file chosen

Now things will get interesting!

3. Click the **Choose File** button, find an image from your computer that has a human face, and then click **Submit**.

After uploading a photo, you should see something like this:



## Google Cloud Platform - Face Detection Sample

This Python Flask application demonstrates App Engine Flexible, Google Cloud Storage, Datastore, and the Cloud Vision API.

Upload File:  No file chosen



Sundar.jpg was uploaded 2017-02-23 19:31:53.766851+00:00.

Joy Likelihood for Face: VERY\_LIKELY

**Note:** When you are done testing your application locally, press CTRL+C in Cloud Shell to shut down the local web server.

Click **Check my progress** below to check your lab progress.

Run the Application

## Task 4. Exploring the code

## Sample code layout

The sample has the following layout:

templates/ homepage.html /\* HTML template that uses Jinja2 \*/ app.yaml /\* App Engine application configuration file \*/ main.py /\* Python Flask web application \*/ requirements.txt /\* List of dependencies for the project \*/

### main.py

This Python file is a Flask web application. The application allows users to submit photos (preferably of faces), which are stored in Cloud Storage and analyzed using the face detection feature of the Cloud Vision API. Key information about each photo is stored in Datastore, Google Cloud's NoSQL database, where it is accessed each time a user visits the website.

This application uses the Google Cloud client libraries for Storage, Datastore, and Vision. These client libraries make it easy to access Cloud APIs from your favorite programming languages.

Let's take a look at some key snippets of the code.

The imports section at the top is where we import the various packages we need for our code. This is how we import our Google Cloud client libraries for Datastore, Storage, and Vision:

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

### Code that directs what happens when a user visits the root URL of the website

Here is the code for what happens when a user visits the root URL of the website. A Datastore client object is created, which is used to access the Datastore client library. A query on Datastore is run for entities of kind Faces. Finally, the HTML template is rendered, passing in the `image_entities` we extract from Datastore as a variable.

```
@app.route('/') def homepage(): # Create a Cloud Datastore client. datastore_client = datastore.Client() # Use the Cloud Datastore client to fetch information from Datastore about # each photo. query = datastore_client.query(kind='Faces') image_entities = list(query.fetch()) # Return a Jinja2 HTML template and pass in image_entities as a parameter. return render_template('homepage.html', image_entities=image_entities)
```

Let's take a look at how [entities](#) are saved to Datastore. Datastore is Google Cloud's NoSQL database solution. Data is stored in objects called *entities*. Each entity is assigned a unique identifying *key*, which can be created using a *kind* and a *key name* string. A *kind* is an organizational bucket for what type of *entity* it is. For example, we might want to set up *kinds* for Photos, People, and Animals.

Each *entity* can have multiple developer-defined *properties*, which can have values of a number of types, including integers, floats, strings, dates, or binary data:

```
# Create a Cloud Datastore client. datastore_client = datastore.Client() # Fetch the current date / time. current_datetime = datetime.now() # The kind for the new entity. kind = 'Faces' # The name/ID for the new entity. name = blob.name # Create the Cloud Datastore key for the new entity. key = datastore_client.key(kind, name) # Construct the new entity using the key. Set dictionary values for entity #
```

```
keys blob_name, storage_public_url, timestamp, and joy. 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 the new entity to Datastore. datastore_client.put(entity)
```

The Storage and Vision client libraries can be accessed programmatically in a similar manner to Datastore. You can open the *main.py* file yourself using *vim*, *emacs*, or *nano* to explore all of the sample code.

## homepage.html

The Flask web framework leverages Jinja2 as a template engine. This allows us to pass in variables and expressions from *main.py* into *homepage.html* that get replaced with values once the page is rendered.

Learn more about Jinja2 at [Template Designer Documentation](#).

This Jinja2 HTML template displays a form for users to submit photos to the database. It also displays each previously submitted image along with its file name, upload date/time, and the likelihood that the face detected by the Vision API is happy.

```
<h1>Google Cloud Platform - Face Detection Sample</h1> <p>This Python Flask application demonstrates App Engine Flexible, Google Cloud Storage, Datastore, and the Cloud Vision API.</p> <br> <html> <body> <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 %}  <p>{{image_entity['blob_name']}} was uploaded {{image_entity['timestamp']}}.</p> <p>Joy Likelihood for Face: {{image_entity['joy']}}</p> {% endfor %} </body> </html>
```

## Task 5. Deploying the App to App Engine Flexible

App Engine Flexible uses a file called *app.yaml* to describe an application's deployment configuration. If this file is not present, App Engine will try to guess the deployment configuration. However, it is a good idea to provide this file.

1. Next, you will modify *app.yaml* using an editor of your choice *vim*, *nano*, or *emacs*. We will use the *nano* editor:

```
nano app.yaml
```

2. Once you have *app.yaml* open, replace `<your-cloud-storage-bucket>` with the name of your Cloud Storage bucket. (If you forgot the name of your Cloud Storage bucket, copy the **Project ID** from the lab details panel).

The *env\_variables* section sets up environment variables that will be used in *main.py* once the application is deployed.

3. Next, set your app to use **manual scaling** by adding this at the end of the file:

```
manual_scaling: instances: 1
```

4. Lastly, change the *python\_version* from 3 to 3.7 to deploy your App Engine successfully.

Your file should look like this:



```
runtime: python env: flex entrypoint: gunicorn -b :$PORT main:app runtime_config: python_version: 3.7
env_variables: CLOUD_STORAGE_BUCKET: <your-cloud-storage-bucket> manual_scaling: instances: 1
```

This is the basic configuration needed to deploy a Python 3 App Engine Flex application. You can learn more about configuring App Engine at [Configuring your App with app.yaml Guide](#).

5. Save and close the file in nano:

- Press CTRL+C.
- At the prompt, type Y and then press ENTER.

6. Update your Cloud Build timeout:

```
gcloud config set app/cloud_build_timeout 1000
```

7. Deploy your app on App Engine by using gcloud:

```
gcloud app deploy
```

If prompted **Do you want to continue (Y/n)**, type Y and then press ENTER.

Watch in Cloud Shell as the application gets built. This will take up to **10** minutes. The App Engine Flexible environment is automatically provisioning a Compute Engine virtual machine for you behind the scenes, and then installing the application, then starting it.

8. After the application is deployed, open the app in your web browser with the following URL:

[https://<PROJECT\\_ID>.appspot.com](https://<PROJECT_ID>.appspot.com) **Note:** If you forgot your **PROJECT\_ID**, run `gcloud config list project` from Cloud Shell.

Click **Check my progress** below to check your lab progress.

Deploy the App

## Congratulations!

You learned how to write and deploy a Python web application to the App Engine Flexible environment!

### Take your next lab

Continue your learning journey with [Firebase Web](#), or try [Deploying Memcached on Kubernetes Engine](#)

### Next steps / Learn more

- Python on Google Cloud: <https://cloud.google.com/python>
- App Engine Flexible Documentation for Python:  
<https://cloud.google.com/appengine/docs/flexible/python>
- Documentation for Python Client Libraries: [Datastore](#), [Storage](#), and [Vision](#)
- Learn more about Client Libraries <https://googlecloudplatform.github.io/google-cloud-python>.
- More Python code samples: <https://cloud.google.com/python/samples>

- Learn more about Flask at <http://flask.pocoo.org>.

## Google Cloud training and certification

...helps you make the most of Google Cloud technologies. **Our classes** include technical skills and best practices to help you get up to speed quickly and continue your learning journey. We offer fundamental to advanced level training, with on-demand, live, and virtual options to suit your busy schedule. **Certifications** help you validate and prove your skill and expertise in Google Cloud technologies.

**Manual Last Updated April 20, 2023**

**Lab Last Tested April 20, 2023**

Copyright 2023 Google LLC All rights reserved. Google and the Google logo are trademarks of Google LLC. All other company and product names may be trademarks of the respective companies with which they are associated.