apps to Python 3 (/appengine/docs/standard/python/migrate-to-python3/).

Getting Started in the Python 2 Standard Environment

This guide is designed to help you learn how to develop and deploy basic Python 2.7 applications that run on the Google App Engine Standard Environment (/appengine/docs/about-the-standard-environment). This guide is intended for use by those new to Google App Engine, its related services, and in particular, using App Engine with the Python language.

Before you begin

Before you can develop an application:

1. Create a new Cloud Console project or retrieve the project ID of an existing project from the Google Cloud Console:

   Go to the Projects page (https://console.cloud.google.com/project)

2. Install and then initialize the Google Cloud SDK (/sdk/docs).

Creating a basic application

This guide uses the Flask web application framework because of its simplicity, ease of use, and extensibility, but the same principles can be applied to any framework that you want to use. This guide teaches you:

- How to create a basic user comment form that will display the content that the user submits via that form on an HTML template that you create.
- How to create a basic application that can serve static files, such as CSS or images.

After you have set up your development environment, you can write the code for the application and deploy it to App Engine.

Basic structure of an app project

This guide uses the following structure for the `flask-app` project:

`flask-app` project structure

- `app.yaml`: Configure the settings of your App Engine application
- `main.py`: Write the content of your application
- `static`: Directory to store your static files
  - `style.css`: Basic stylesheet that formats the look and feel of your template files
- `templates`: Directory for all of your HTML templates
  - `form.html`: HTML template to display your form
  - `submitted_form.html`: HTML template to display the content of a submitted form

Setting up libraries to enable development

This tutorial puts a copy of the Flask library in your app's directory. Note that although the App Engine Python 2.7 runtime environment has a bundled (/appengine/docs/standard/python/tools/built-in-libraries-27) Flask library, that bundled library is an older version that might not work with this tutorial.

To set up the required libraries:

1. Create a file named `appengine_config.py` in the root project directory. When you deploy your application, you can use this file to specify where App Engine should look for third-party libraries:

   appengine/standard/flask/tutorial/appengine_config.py (https://github.com/GoogleCloudPlatform/python-docs-samples/blob/master/appengine/standard/flask/tutorial/appengine_config.py)

ɲ/python-docs-samples/blob/master/appengine/standard/flask/tutorial/appengine_config.py)

```
from google.appengine.ext import vendor

# Add any libraries installed in the "lib" folder.
vendor.add('lib')
```

2. Create a file named `requirements.txt` in the root project directory:

   appengine/standard/flask/tutorial/requirements.txt (https://github.com/GoogleCloudPlatform/python-docs-samples/blob/master/appengine/standard/flask/tutorial/requirements.txt)

:form/python-docs-samples/blob/master/appengine/standard/flask/tutorial/requirements.txt)

```
Flask==1.1.1
Werkzeug==0.16.1
```

3. To run this app on your local computer, you'll need a Python development environment set up, including Python, `pip`, and `virtualenv`. For instructions, refer to Setting Up a Python Development Environment (/python/setup) for Google Cloud Platform.

4. Install dependencies using `virtualenv`:

   Mac OS / LinuxWindows (#windows)

   a. Create an isolated Python environment in a directory external to your project and activate it:

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

   At the end of the tutorial, you can exit your `virtualenv` by typing `deactivate`.

   b. Navigate to your project directory and install dependencies:

```
cd YOUR_PROJECT
pip install -t lib -r requirements.txt
```

The `-t lib` flag copies the libraries into a `lib` folder, which is uploaded to App Engine during deployment. See Using pip requirements.txt with copied libraries (/appengine/docs/standard/python/tools/using-libraries-python-27#pip_requirements) for more information on copying third-party libraries.

The `-r requirements.txt` flag tells `pip` to install everything from a `requirements.txt` file.

`pip` version 6.0.0 or higher is required to install libraries by using this command.

ing: If you are using Homebrew (http://brew.sh/) Python on macOS, you might encounter an tion when running **`pip install -t`**. This problem is related to a known issue s://github.com/Homebrew/brew/blob/master/docs/Homebrew-and-Python.md#note-on-pip-in) Homebrew's configuration of Python. The issue description has a workaround.

Creating the app.yaml file

rtant: The **`app.yaml`** file must be located in your application's root directory.

You can configure your App Engine application's settings in the **`app.yaml`** file that you create manually or as part of the creation of your development project. The **`app.yaml`** file is a configuration file that tells App Engine how to run your application and how to map URLs to static files and Python modules.

To create the **`app.yaml`** file:

1. Create a file named **`app.yaml`** in the root directory of your project.
2. Add the following lines to the file:

appengine/standard/flask/tutorial/app.yaml (https://github.com/GoogleCloudPlatform/python-docs-samples/blob/master/appengine/standard/flask/tutorial/app.yaml)

oudPlatform/python-docs-samples/blob/master/appengine/standard/flask/tutorial/app.yaml)

```
runtime: python27
api_version: 1
threadsafe: true

libraries:
- name: ssl
  version: latest

handlers:
- url: /static
  static_dir: static
- url: /.*
  script: main.app
```

More reference information about the `app.yaml` file can be found in the app.yaml reference documentation (/appengine/docs/standard/python/config/appref).

Creating a request handler for your Flask app

When App Engine receives a web request for your application, it calls the handler script that corresponds to the URL, as described in the application's `app.yaml` configuration file. The Python 2.7 runtime supports the WSGI standard (http://www.wsgi.org/). WSGI is preferred, and some features of Python 2.7 do not work without it. The configuration of your application's script handlers determines whether a request is handled using WSGI.

The server determines which Python application object to call by comparing the URL of the request to the URL patterns in the app's configuration file. It then calls the application object using the arguments as defined in the WSGI standard (http://www.wsgi.org/). The application object performs actions appropriate to the request, then prepares a response and returns it as a list of strings.

The following request handlers take the information that is submitted in the form in the `/templates/form.html` file and places that information onto the

`/templates/submitted_form.html` template file:

1. Create a new file called `main.py` in the root directory of your application.
2. Import the Flask framework and the Flask interfaces that you want to use:

appengine/standard/flask/tutorial/main.py (https://github.com/GoogleCloudPlatform/python-docs-samples/blob/master/appengine/standard/flask/tutorial/main.py)

```
CloudPlatform/python-docs-samples/blob/master/appengine/standard/flask/tutorial/main.py)
```

```
from flask import Flask, render_template, request
```

3. Add this line to create an instance of the `Flask` class and assign it to a variable called `app`:

appengine/standard/flask/tutorial/main.py (https://github.com/GoogleCloudPlatform/python-docs-samples/blob/master/appengine/standard/flask/tutorial/main.py)

```
CloudPlatform/python-docs-samples/blob/master/appengine/standard/flask/tutorial/main.py)
```

```
app = Flask(__name__)
```

4. Create a request handler that displays a form using the `form.html` template:

appengine/standard/flask/tutorial/main.py (https://github.com/GoogleCloudPlatform/python-docs-samples/blob/master/appengine/standard/flask/tutorial/main.py)

```
CloudPlatform/python-docs-samples/blob/master/appengine/standard/flask/tutorial/main.py)
```

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

When the user navigates to the `/form/` directory within the application, the `form.html` template that you will create will be displayed.

5. Create a request handler that handles the information from the submitted form:

appengine/standard/flask/tutorial/main.py (https://github.com/GoogleCloudPlatform/python-docs-samples/blob/master/appengine/standard/flask/tutorial/main.py)

CloudPlatform/python-docs-samples/blob/master/appengine/standard/flask/tutorial/main.py)

```
@app.route('/submitted', methods=['POST'])
def submitted_form():
    name = request.form['name']
    email = request.form['email']
    site = request.form['site_url']
    comments = request.form['comments']
```

The application stores the form information in the variables that you created here. These variables will allow you to post the data from the form onto the `submitted_form.html` template that you will create.

You can learn more about how to quickly get started with Flask in the Flask quickstart guide (http://flask.pocoo.org/docs/0.12/quickstart/).

You can easily extend the functionality of this form. For example, you can use the Mail API (/appengine/docs/standard/python/mail), Mailgun (/appengine/docs/standard/python/mail/mailgun), Mailjet (/appengine/docs/standard/python/mail/mailjet), or SendGrid (/appengine/docs/standard/python/mail/sendgrid) to send the comments that users submit to yourself or to others.

Setting up Jinja2 templates

Since HTML that is embedded in code can be difficult to maintain, you should use a templating system, storing HTML in a separate file that uses special syntax to specify where the data returned from an application appears. You can use your template engine of choice by bundling it with your application code. For your convenience, App Engine includes the Django

(https://docs.djangoproject.com/en/dev/topics/templates/) and Jinja2 (http://jinja.pocoo.org/docs/) templating engines.

1. Add the following line to the end of the `submitted_form()` function:

   appengine/standard/flask/tutorial/main.py (https://github.com/GoogleCloudPlatform/python-docs-samples/blob/master/appengine/standard/flask/tutorial/main.py)

   ```
   return render_template(
       'submitted_form.html',
       name=name,
       email=email,
       site=site,
       comments=comments)
   ```

   This line uses the `render_template()` interface to render the `submitted_form.html` template with submitted form information.

2. Create the `form.html` and `submitted_form.html` templates:

   a. Create a new folder called `templates` in your root directory:

   ```
   mkdir templates
   ```

   b. Create `form.html` in the `templates` directory of your project:

   appengine/standard/flask/tutorial/templates/form.html (https://github.com/GoogleCloudPlatform/python-docs-samples/blob/master/appengine/standard/flask/tutorial/templates/form.html)

   ```
   <html>
     <head>
       <title>Submit a form</title>
   ```

```
    <link rel="stylesheet" type="text/css" href="/static/style.
  </head>
  <body>
    <div id="container">
      <div class="pagetitle">
        <h1>Submit a form</h1>
      </div>
      <div id="main">
        <form method="post" action="{{ url_for('submitted_form
          <label for="name">Name:</label>
          <input type="text" name="name"><br />
          <label for="email">Email address:</label>
          <input type="email" name="email"><br />
          <label for="site_url">Website URL:</label>
          <input type="url" name="site_url"><br />
          <label for="comments">Comments:</label>
          <textarea name="comments"></textarea><br />
          <input type="submit">
        </form>
      </div>
    </div>
  </body>
</html>
```

c. Create **`submitted_form.html`** in the **`templates`** directory of your project:

appengine/standard/flask/tutorial/templates/submitted_form.html (https://github.com/GoogleCloudPlatform/python-docs-samples/blob/master/appengine/standard/flask/tutorial/templates/submitted_form.html)

```
<html>
 <head>
   <title>Submitted form</title>
   <link rel="stylesheet" type="text/css" href="/static/style.
 </head>
 <body>
   <div id="container">
     <div class="pagetitle">
       <h1>Form submitted</h1>
     </div>
```

```
    <div id="main">
      <p>Thanks for your submission, {{name}}!</p>
      <p>Here's a review of the information that you sent:</p
      <p>
         <strong>Name</strong>: {{name}} <br>
         <strong>Email</strong>: {{email}} <br>
         <strong>Website URL</strong>: {{site}} <br>
         <strong>Comments</strong>: {{comments}}       </p>
    </div>
   </div>
 </body>
</html>
```

To learn more about using templates with Flask and Jinja2, visit the official Flask documentation (http://flask.pocoo.org/docs/0.12/templating/).

Serving static files

Serving static files is more efficient for some content, such as images, CSS or Flash animations, which aren't generated dynamically when a page is requested.

Create a CSS file and create a handler for it:

1. Create a new folder called `static` in your root directory:

   ```
   mkdir static
   ```

2. Create the `style.css` file that will modify the look of your template files that you just created. Create the file in the `static` folder of your project and add the following styling:

   appengine/standard/flask/tutorial/static/style.css (https://github.com/GoogleCloudPlatform/python-docs-samples/blob/master/appengine/standard/flask/tutorial/static/style.css)

atform/python-docs-samples/blob/master/appengine/standard/flask/tutorial/static/style.css)

```
.pagetitle {
    color: #800080;
```

```
}
```

3. The `app.yaml` file you created earlier specifies the `static` directories that contain static files:

   appengine/standard/flask/tutorial/app.yaml (https://github.com/GoogleCloudPlatform/python-docs-samples/blob/master/appengine/standard/flask/tutorial/app.yaml)

oudPlatform/python-docs-samples/blob/master/appengine/standard/flask/tutorial/app.yaml)

```
handlers:
- url: /static
  static_dir: static
- url: /.*
  script: main.app
```

   The `handlers` section defines two handlers for URLs. When App Engine receives a request for a URL beginning with `/static`, it maps the remainder of the path to files in the `static` directory, and if an appropriate file is found, the contents of the file are returned to the client.

For more information on URL mapping and other options you can specify in `app.yaml`, see the `app.yaml` reference (/appengine/docs/standard/python/config/appref#handlers_element).

Test the application

Test the application using the local development server (`dev_appserver.py`), which is included with the SDK.

1. From within the root directory where the app's app.yaml (/appengine/docs/standard/python/config/appref) configuration file is located, start the local development server with the following command:

```
dev_appserver.py app.yaml
```

The local development server is now running and listening for requests on port 8080.

Something go wrong? (#test_the_application)

a. Visit http://localhost:8080/form (http://localhost:8080/form) in your web browser to view the app.

Make a change

You can leave the development server running while you develop your application. The development server watches for changes in your source files and reloads them if necessary.

1. Try it now: Leave the development server running, then edit `templates/form.html` to change the `Submit a form` text within the `<h1>` tags to something else.
2. Reload http://localhost:8080/form (http://localhost:8080/form) to see the results.

Deploying your application

To upload the app, run the following command from within the root directory of your project where the `app.yaml` file is located:

```
ud app deploy
```

Optional flags:

- Include the `--project` flag to specify an alternate Cloud Console project ID to what you initialized as the default in the `gcloud` tool. Example: `--project [YOUR_PROJECT_ID]`
- Include the `-v` flag to specify a version ID, otherwise one is generated for you. Example: `-v [YOUR_VERSION_ID]`

To learn more about deploying your app from the command line, see Deploying a Python App (/appengine/docs/standard/python/tools/uploadinganapp).

Viewing your application

Open your browser and view your app at `https://PROJECT_ID.REGION_ID` (#appengine-urls)`.r.appspot.com/form`.

Support

If you encounter problems during the development of your application, you can get help from technical support and developer communities (/support).

What's next

- Test the quality of your code and improve your development processes (/appengine/docs/standard/python/tools/localunittesting).
- View the logs for your application and make sense of the information listed in the log (/appengine/docs/standard/python/logs).
- Learn more about the App Engine Standard Environment (/appengine/docs/about-the-standard-environment).
- Explore the App Engine Python API and reference documentation (/appengine/docs/standard/python/apis).
- Learn how to retrieve, verify, and store user credentials with server-side Firebase Authentication (/appengine/docs/standard/python/authenticating-users-firebase-appengine).

Last updated 2020-02-18.