Setting up Flask applications on PythonAnywhere

There are two main ways to set up a Flask application on PythonAnywhere:

1. Starting from scratch using our default versions of Flask
2. Importing a pre-existing app using Manual configuration, and using a virtualenv

The first option works well if you're just playing around and want to throw something together from scratch. Go to the Web Tab (https://www.pythonanywhere.com/web_app_setup) and hit **Add a new Web App**, and choose Flask and the Python version you want.

The second option is described in more detail below

Getting your code onto PythonAnywhere

This guide assumes you've already managed to get your code onto PythonAnywhere. Check out the uploading and downloading files (/pages/UploadingAndDownloadingFiles) guide if you need to.

For the purposes of these examples, we'll assume your code lives at `/home/yourusername/mysite`

Check your config

If you're importing existing code, review all of your Flask configuration settings to ensure that they match their new home. For instance, if you've specified a SERVER_NAME in your config, make sure that it matches the web app name.

Setting up your virtualenv

Open up a new Bash console from your Dashboard (https://www.pythonanywhere.com/consoles) and run

```
mkvirtualenv --python=/usr/bin/python3.6 my-virtualenv  # use whichever p
pip install flask
```

You'll see the prompt changes from a `$` to saying `(my-virtualenv)$` -- that's how you can tell your virtualenv is active. Whenever you want to work on your project in the console, you need to make sure the virtualenv is active. You can reactivate it at a later date with

```
$ workon my-virtualenv
(my-virtualenv)$
```

You can also install any other dependencies you may have at this point, like Sqlalchemy, using `pip install flask-sqlalchemy` , or `pip install -r requirements.txt` , if you have a requirements.txt

Setting up the Web app using Manual configuration

Go to the Web Tab (https://www.pythonanywhere.com/web_app_setup) and hit **Add a new web app**. Choose **Manual Configuration**, and then choose the **Python version** -- make sure it's the same version as the one you used in your virtualenv

Now go to the **Virtualenv** section, and enter your virtualenv name: *my-virtualenv*. When you hit enter, you'll see it updates to the full path to your virtuaelenv (*/home/yourusername/.virtualenvs/my-virtualenv*).

Finally, go edit the wsgi configuration file. You'll find a link to it near the top of the Web tab.

Configuring the WSGI file

To configure this file, you need to know which file your flask app lives in. The flask app usually looks something like this:

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

Make a note of the path to that file, and the name of the app variable (is it "app"? Or "application"?) -- in this example, let's say it's `/home/yourusername/mysite/flask_app.py` , and the variable is "app".

In your WSGI file, skip down to the flask section, uncomment it, and make it looks something like this:

```
import sys
path = '/home/yourusername/mysite'
if path not in sys.path:
    sys.path.insert(0, path)

from flask_app import app as application
```

Do not use app.run()

When you're using Flask on your own PC, you'll often "run" flask using a line that looks something like this:

```
app.run(host='127.0.0.1',port=8000,debug=True)
```

That won't work on PythonAnywhere -- the only way your app will appear on the public internet is if it's configured via the web tab, with a wsgi file.

More importantly, '**if app.run() gets called when we import your code, it will crash your app**', and you'll see a 504 error on your site, as detailed in Flask504Error (/pages/Flask504Error)

Thankfully, most Flask tutorials out there suggest you put the `app.run()` inside an `if __name__ = '__main__':` clause, which will be OK, because that won't get run when we import it.

This is ok

```
app = Flask(__name__)

@app.route('/')
def home():
    # etc etc, flask app code

if __name__ == '__main__':
    app.run()
```

This is not ok:

```
app = Flask(__name__)
@app.route('/')
def home():
    # etc etc, flask app code

app.run()
```

What about my database config?

Many guides on the Internet also suggest you put your database setup inside the `__main__` clause, like this:

```
if __name__ == '__main__':
    db.create_all()
    app.run()
```

That will work fine on your machine, but, again, we don't want to use app.run() on PythonAnywere (https://www.pythonanywhere.com/). But you'll still want to be able to run `db.create_all()` every so often on PythonAnywhere, to update your database tables or whatever it may be.

Two solutions -- either just run it from a Bash console (remembering to activate your virtualenv first) and then Ctrl+C the flask server when it runs

```
$ workon my-virtualenv
(my-virtualenv)$ python flask_app.py
 * Running on http://127.0.0.1:5000/
^C
(my-virtualenv)$
```

Or make a clever little if in your main that checks if it's running on PythonAnywhere, eg:

```
from socket import gethostname
[...]

if __name__ == '__main__':
    db.create_all()
    if 'liveconsole' not in gethostname():
        app.run()
```

Want to improve this page? Submit a pull request! (https://github.com/pythonanywhere/help_pages)