Create a Simple Flask Application with Cloud9, Heroku and MongoDB

Douglas Starnes

Create a Simple Flask Application with Cloud9, Heroku and MongoDB

Douglas Starnes

This book is for sale at http://leanpub.com/flask-cloud9-heroku-mongodb-ebook

This version was published on 2015-05-22

Contents

Introduction

Deploying a Flask app to Heroku and MongoLab with Cloud9

This ebook was inspired by a series of presentations that I have been doing at the Memphis Python User Group (MEMpy). I had some requests for a written copy of the materials. These are being posted to Github[1] but I decided to format them as an ebook because the README in the Github repository is getting very long.

[1] http://github.com/douglasstarnes/mempy-flask-heroku

Services Used in this Book

What you'll need

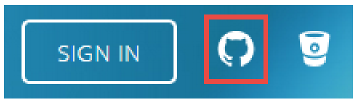
Before you begin you'll need to sign up for a few online services:

- Github - If you haven't created a Cloud9 account yet, you can easily do so with Github
- Cloud9 - Use your Github account to log in
- Heroku - This is the cloud hosting service
- MongoLab - This is the database

Github The home page for Github is github.com[2]. We won't be using Github for in this tutorial for anything other than creating a Cloud9 account. If you already have a Github account, skip to the next section. Otherwise, go to the Github home page and create a new account. You'll have to verify your email address by clicking on a link in an email they will send to you. After that, you're ready to move on to Cloud9.

Cloud9 IDE This is the browser-based development environment we will be using. It's completely free to use except there is one note of caution I like to reiterate. **In the free Cloud9 workspaces, all code is public read-only**. This means that you should not store sensitive data like OAuth tokens or API keys in the code (and you shouldn't do this anyway). We'll see a better way of doing this when we get to MongoLab.

The Cloud9 home page is c9.io[3]. You can use your Github account (see the previous section) to create a Cloud9 account. Just click on the icon that looks like the silhouette of a cat next to the 'SIGN IN' button:



Sign in to Cloud9 with Github

If you already have a Cloud9 account associated with Github you'll be taken to the dashboard. Otherwise, a screen will ask you to authorize Cloud9 to use your Github account. Once you agree, you'll be taken to the dashboard.

[2]http://github.com
[3]http://c9.io

Heroku Heroku is a cloud web application hosting provider. We will use Heroku to serve our application. Heroku is free just like Github and Cloud9 (or at least it has a free plan which will be more than enough for our simple application.) but you can't use Github to log in. So you'll need to go to the Heroku homepage at heroku.com[4] and create an account. The process is similar to Github and you'll need to verify your email by clicking on a link they will send you in an email.

One note about Heroku, it also has paid plans. By default, you get 5 applications and a limited number of add on services (such as databases) for free. However, if you verify your account with a credit card, they will give you 100 application and more add ons. This will make the next step, adding MongoDB support, very easy. However, I understand that not everyone will want to do this so I have included the instructions for using MongoLab with Heroku without a verified account.

MongoLab MongoLab is the host for the database that we will use. This database is called MongoDB. If you haven't worked with MongoDB or it's relatives (collectively referred to as NoSQL) it's quite different from other databases you might have used in the past. However, it's super simple to use, is pre-installed on Cloud9 for testing purposes, has an excellent Python library and is free! So just like Heroku, you can't use Github to login. You'll need to go to the MongoLab home page at mongolab.com[5] and create an account. After that, head back to Cloud 9 and we'll start coding!

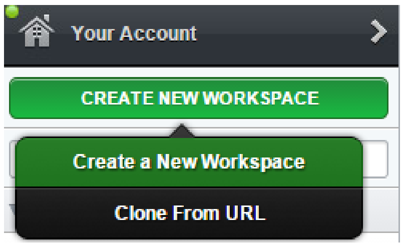
[4]http://heroku.com

[5]http://mongolab.com

Cloud9 IDE

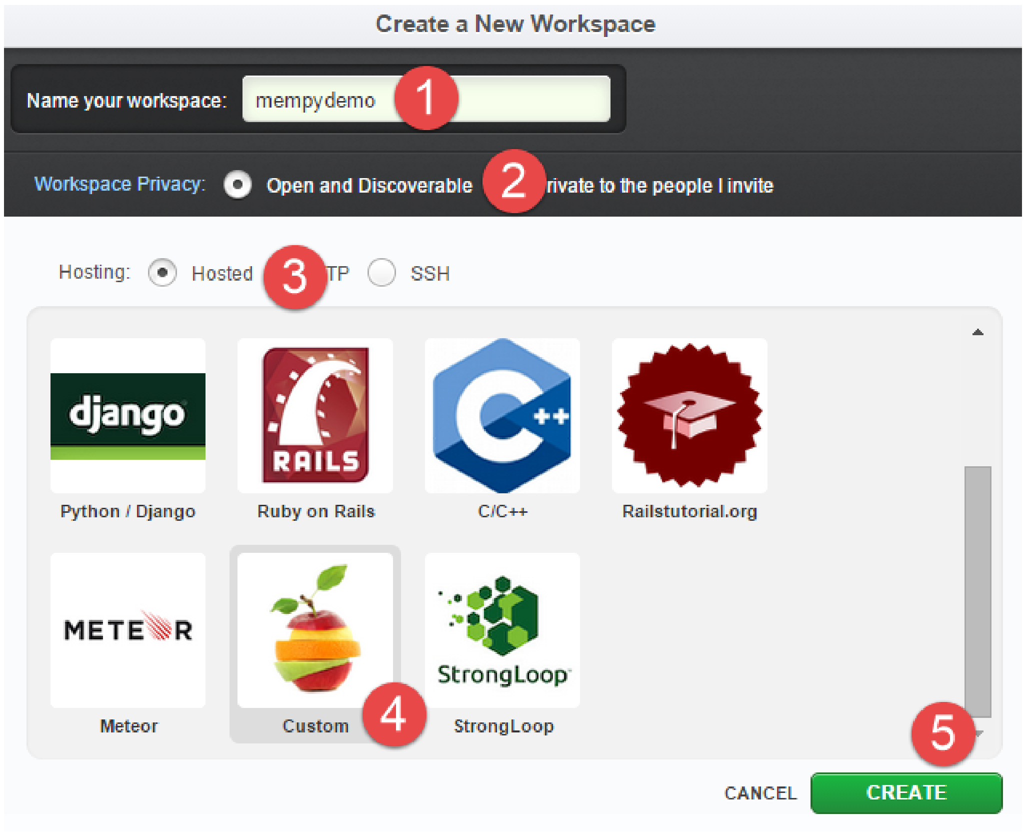
Setting up Cloud9

There is not a lot to do here because most everything is preinstalled but there are a few things we need to do. First we have to create a new workspace. When you log in to Cloud9, you'll be taken to the dashboard which has a list of your workspaces on the left. By default, Cloud9 creates a demo project workspace for you, but we are going to create one from scratch. So click the green 'CREATE NEW WORKSPACE' button in the upper left and then on the 'Create a New Workspace' link in the pop up menu:



Create a new Cloud9 workspace

You'll be presented with the 'Create a New Workspace' dialog:


Create a New Workspace

Name your workspace: mempydemo

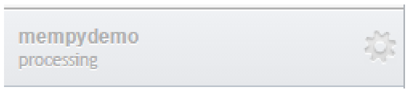
Workspace Privacy: (●) Open and Discoverable ( ) Private to the people I invite

Hosting: (●) Hosted ( ) FTP ( ) SSH

Create a New Workspace

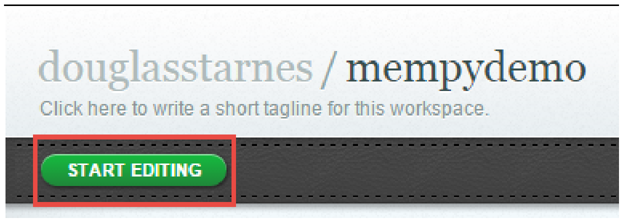
1. Give your workspace a name. This name does not have to be unique across Cloud9, only to your account so you can use *mempydemo* like shown above.
2. Select *Open and Discoverable* for *Workspace Privacy*. This is the option to get free workspaces.
3. For *Hosting* select *Hosted*. Again, this is for the free workspaces.
4. Select *Custom* for the workspace type. This will give us a blank workspace without any opinions as to the framework or language we will be using.
5. Click the green *CREATE* button.

A new entry in the left of the window will come up with your workspace name and a gear. This mean Cloud9 is creating the workspace. It should only take a few seconds. Cloud9 is very fast!


mempydemo
processing

Workspace pending

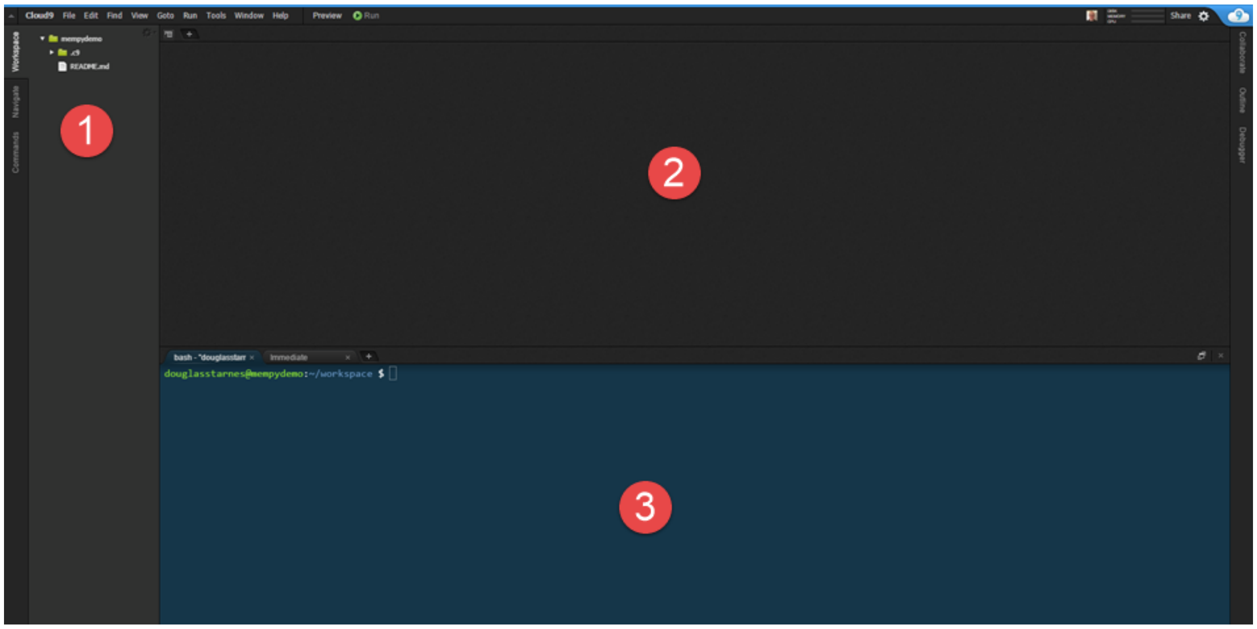
When Cloud9 is finished you can click the green *START EDITING* button in the right side of the window.


douglasstarnes / mempydemo

Click here to write a short tagline for this workspace.

Begin editing

Below is a picture of the default layout for the Cloud9 IDE:



```
douglasstarnes@mempydemo:~/workspace $
```

C9 workspace

1. The project manager that shows a tree view of the files in this workspace.
2. An editor similar to Sublime Text with optional vim binding.
3. A Linux terminal with sudo shell access.

There are only a few more things we need to do to get set up and all of them can be done from the terminal.

First, since all new Python development is going to be done in Python 3, we should use it. However, the default version of Python installed on Cloud9 workspaces (which are running on top of Ubuntu 14.04) is 2.7.6 (feel free to verify this by running `python --version`. So we are going to create a *virtual environment* which is a special instance of Python that thinks it is the default version. Virtual environments can have their own Python version and implementation as well as their own set of libraries installed. Fortunately, Cloud9 includes the virtual environment scripts for us. So to create a virtual environment with Python 3, run the following command in the terminal:

```
mkvirtualenv --python=`which python3` mempydemo
```

You'll see some output similar to this:

```
Running virtualenv with interpreter /usr/bin/python3
Using base prefix '/usr'
New python executable in mempydemo/bin/python3
Also creating executable in mempydemo/bin/python
Installing setuptools, pip...done.
```

The last parameter is the name of the virtual environment. You can use any name you want but since this is specific only to this workspace, you can use *mempydemo* to make it easier to follow along. Notice that the prompt in the terminal is now prefixed with the name of the virtual environment in parentheses. This means you are inside a virtual environment. To verify this run the command `python --version` again and you should see a variant of Python 3, 3.4.0 as of this writing.

To leave a virtual environment, run the command `deactivate`. The prefix willbe removed from the prompt and Python 2.7.6 will be the default version again. Again, check this with `python --version`. To enter the virtual environment, run `workon mempydemo` (or whatever you named your virtual environment) and you'll see the environment is now active again.

If you look at the output from `mkvirtualenv` the last line says it installed an application called `pip`. This is a package manager that will retrieve and install Python packages and their dependencies from the internet. We are going to use it to install Flask, the web framework we will use to write our application. Run this command in the terminal:

```
pip install Flask
```

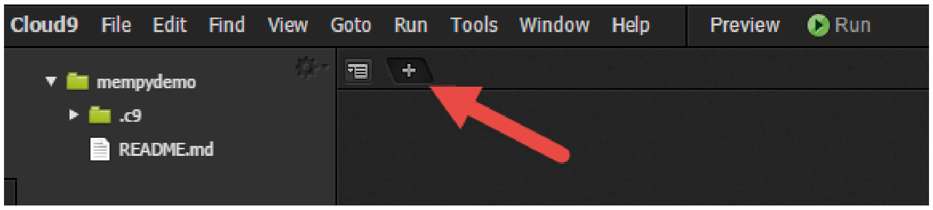
This will create a lot of output and you don't need to worry about it as long as there are no errors. However, notice that at the end of the output it installed Flask as well as four other packages. These are packages that Flask depends on. Also, just before that, pip compiled a native C extension that one of the packages uses. Cloud9 is set up for you to do this. Otherwise, you would have had to install some prerequisite development libraries. Cloud9 helps you out more than you think!

We'll install some more packages later but this will get us started. Let's go ahead an write a simple application.

Flask Microframework

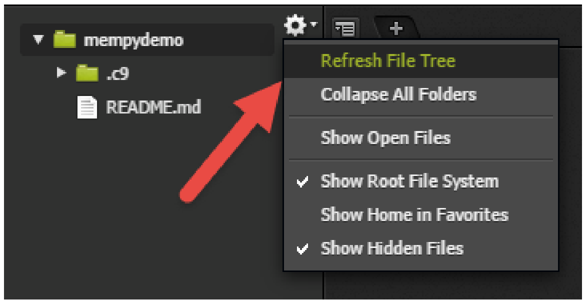
Our first application

We need to create a new Python file to hold our code. This can be done in many ways. You can create a new file through the IDE by selecting *File* -> *New File* from the menu, by right clicking on a folder in the project manager and by click the plus (+) tab in the editor:



The plus tab

My preferred way is to create create new files in the terminal. To do this make sure you are in the *workspace* directory in your home directory (`cd ~/workspace` to move there) and then run `touch main.py` where *main.py* is the name of the file to create. Then double click on that file in the project manager. If you don't see it you may have to refresh the file tree which you can do by clicking on the gear in the project manager and selecting *Refresh File Tree*.



Refresh File Tree

The following code will comprise our application:

```python
from flask import Flask
import os

app = Flask(__name__)
app.debug = True

@app.route('/')
def index():
    return 'Hello MEMpy'

if __name__ == '__main__':
    port = int(os.getenv('PORT', 8080))
    host = os.getenv('IP', '0.0.0.0')
    app.run(port=port, host=host)
```

We'll go more in depth with Flask later but for a quick look at the code. The first two lines import packages, namely Flask and os which we use in the entry point to get values for environment variables. The next two lines create an instance of Flask and turn on debugging so that if (when) errors happen we will see a nice stack trace in the browser. (Of course this should **NEVER** be enabled in a production application.) The `route()` decorator tells Flask that when it receives a request for the root of the site to invoke the `index()` function. That function merely returns a string which Flask will wrap in an HTTP response and send back to the browser. The entry point extracts values for environment variables named *PORT* and *IP*. This is because if the application is started on a certain port and ip address that Cloud9 has reserved, the application will be accessible to the public web via a special URL. These are stored in the environment variables. The last line starts Flask on the port and ip we retrieved before.

To start this app, just run `python main.py` in the terminal. You should see the following output:

```
* Running on http://0.0.0.0:8080/ (Press CTRL+C to quit)
* Restarting with stat
```

The first line tells us the server did indeed start. And the second line is because it is running in debug mode.

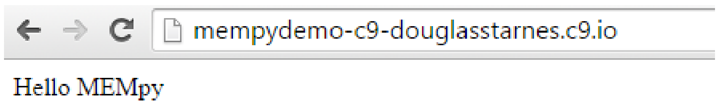
Now let's access our application from a browser. The URL we can go to for our applcation is the form:

```
[workspace]-c9-[username].c9.io
```

For example, my workspace name is *mempydemo* and my username is *douglasstarnes* so the URL for my application would be

```
http://mempydemo-c9-douglasstarnes.c9.io
```

Going to the URL for the application should yield the message 'Hello MEMpy'.



Hello MEMpy

Hello MEMpy

To stop the server press *Ctrl-C* in the terminal.

Obviously, this is not intended for long term hosting. Also, after you close the browser, the workspace will become idle and the process running the server will be terminated. For a robust hosting solution, we'll turn to Heroku.