Deploy Django App to Heroku

Why Heroku?

Heroku is one of the longest running and popular cloud-based PaaS services. It originally supported only Ruby apps, but now can be used to host apps from many programming environments, including Django!

> We are choosing to use Heroku for several reasons:

- Heroku has a free tier that is really free (albeit with some limitations).
- As a PaaS, Heroku takes care of a lot of the web infrastructure for us. This makes it much easier to get started, because you don't worry about servers, load balancers, reverse proxies, or any of the other web infrastructure that Heroku provides for us under the hood.
- Mostly it just works, and if you end up loving it, scaling your app is very easy.

Limits

- App names cannot exceed 30 characters.
- Unverified accounts can create at most 5 apps. Verified accounts can create no more than 100 apps.
- A single app can have up to 1,000 custom domains assigned to it.
- Heroku only stores the last 1500 lines of log history. If you'd like to persist more than 1500 lines, use a logging add-on or create your own syslog drain.
- You can have up to 5 drains on any given app.
- More Info

How does Heroku work?

- Heroku runs Django websites within one or more "Dynos", which are isolated, virtualized Unix containers that provide the environment required to run an application
- For Django apps we provide this information in a number of text files:

- requirements.txt: the Python component dependencies, including Django.
- Procfile: A list of processes to be executed to start the web application. For Django this will usually be the Gunicorn web application server (with a .wsgi script).
- wsgi.py: WSGI configuration to call our Django application in the Heroku environment.

Usage

- If you don't have git installed, follow this Tutorial and come back here.
- Make a copy of your project or use a seperate git branch.
- Make sure your virtual environment is activated.
- Add your dependencies to requirements.txt by typing in the terminal,

```
pip freeze > requirements.txt
```

Configuring Django Apps for Heroku

- First, and most importantly, Heroku web applications require a Procfile. This file is used to explicitly declare your application's process types and entry points. It is located in the root of your repository.

> > Procfile

```
release: python manage.py makemigrations --no-input
release: python manage.py migrate

web: gunicorn myproject.wsgi
```

- This Procfile requires Gunicorn, the production web server that we recommend for Django applications. For more information, see Deploying Python Applications with Gunicorn.
- Installing gunicorn

```
pip install gunicorn
```

> > Be sure to add gunicorn to your requirements.txt file as well.

- Django settings for static assets can be a bit difficult to configure and debug. However, if you just add the following settings to your settings.py, everything should work exactly as expected:

```
BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))

# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/1.9/howto/static-files/
```

```
STATIC_ROOT = os.path.join(BASE_DIR, 'staticfiles')
STATIC_URL = '/static/'

# Extra places for collectstatic to find static files.
STATICFILES_DIRS = (
    os.path.join(BASE_DIR, 'static'),
)
```

- Django does not support serving static files in production. However, the fantastic WhiteNoise project can integrate into your Django application, and was designed with exactly this purpose in mind.
- Installing Whitenoise

```
pip install whitenoise
```

> Be sure to add whitenoise to your requirements.txt file as well.

- Install WhiteNoise into your Django application. This is done in settings.py's middleware section (at the top):

```
MIDDLEWARE_CLASSES = (
    # Simplified static file serving.
    # https://warehouse.python.org/project/whitenoise/
    'whitenoise.middleware.WhiteNoiseMiddleware',
    ...
```

- Finally, if you'd like gzip functionality enabled, also add the following setting to settings.py.

```
# Simplified static file serving.
# https://warehouse.python.org/project/whitenoise/

STATICFILES_STORAGE =
'whitenoise.storage.CompressedManifestStaticFilesStorage'
```

- Installing django-heroku:

```
pip install django-heroku
```

> Be sure to add gunicorn to your requirements.txt file as well.

- Add this in settings.py

```
import django_heroku
```

```
# Activate Django-Heroku.
django_heroku.settings(locals())
```

- Make a Heroku account
- Download Heroku CLI
- Configure Django Heroku
- If collectstatic failed during a build, a traceback was provided that will be helpful in diagnosing the problem. If you need additional information about the environment collectstatic was run in, use the DEBUG_COLLECTSTATIC configuration.

```
heroku config:set DEBUG_COLLECTSTATIC=1
```

- Sometimes, you may not want Heroku to run collectstatic on your behalf. You can disable the collectstatic build step with the DISABLE_COLLECTSTATIC configuration:

```
heroku config:set DISABLE_COLLECTSTATIC=1
```

** PS: if Heroku isn't recognized as a command, please close your terminal and editor and then re-open it.

- DEBUG = False in settings.py
- ALLOWED_HOSTS = ['your_app_name.herokuapp.com', 'localhost', '127.0.0.1'] in settings.py
- DISABLE_COLLECTSTATIC = 1