Last Updated: March 02, 2016 · bt3gl 

Deploying a Flask App at Heroku

HEROKU FLASK GIT PYTHON GUNICORN

I was playing with Flask and I wrote a simple parody: the Anti-Social Network. The Heroku platform is very flexible and it supports several programming languages. To deploy an application to Heroku, you use Git to push the application to Heroku's server.

Running in a Production Server

Heroku does not provide a web server but it expects it to start their own servers and listen on the port number set in environment variable PORT. Flask will perform very poorly because it was not designed to run in a production environment. To ameliorate this, you may use a production-ready web server such as Gunicorn.

```
$ pip install gunicorn
```

Now, run your app with:

```
$ gunicorn manage:app
```

Gunicorn uses port 8000 instead of 5000.

Heroku Setting Up

Create an account at Heroku.com

If you haven't done it yet. Remember: you will be able to keep up to five applications running (you can always delete them if you need).

Install Git and Heroku Toolbelt

You can find instructions at Heroku.com.

For example, if you are in an AWS EC2 Ubuntu instance, you can use:

```
$ sudo apt-get install -y git-core
$ wget -qO- https://toolbelt.heroku.com/install-ubuntu.sh | sh
```

You can check if it worked with:

```
$ which git
$ which heroku
```

Now, login at Heroku:

```
$ heroku login
```

Authorize your Machine at Heroku

Create and add a SSH Key at Heroku:

```
$ ssh-keygen -t rsa
$ heroku keys:add
```

The public and private keys will be at ~/.ssh. I recommend always backup your keys. In addition, never share your private key.

Creating a Git Repository

Heroku's push/commits work just like as Git. But instead of using the "origin" you use "heroku" (you can verify this later at .git/refs/remotes/). In other words, your project's control version (development) is done by using:

```
$ git push origin master (or any branch you like)
```

and the deployment at Heroku (production) is done using:

```
$ git push heroku master (or any branch you like)
```

In the root of your project, go ahead and create a Git repository, commit, add, push:

```
$ git init
$ git add -A
$ git commit -m "First commit"
$ git push origin master
```

The Heroku Repository

Creating an App

Now, let's create our app at Heroku:

```
$ heroku create <app-name>
```

You can check all your current applications with:

```
$ heroku apps
```

Addons and Environment Variables

Now it's time to add the addons and the environment variables to your app at the Heroku server. For the app I mentioned in the beginning, I type:

```
$ heroku addons:add heroku-postgresql:dev
$ heroku pg:promote HEROKU_POSTGRESQL_ONYX_URL

$ heroku config:set MAIL_USERNAME="<login>"
$ heroku config:set MAIL_PASSWORD="<password>"
```

You can always check your configuration with:

```
$ heroku config
```

Adding Requirements

Heroku needs to know what libraries and packages it needs to install to be able to run your application. For this, create a file requirements.txt in the root of your app, with all the libraries from your environment. One way of doing this is by:

```
$ cat pip freeze >> requirements.txt
```

Adding Procfile

Next, Heroku needs to know the command to use to start your app. This is given by a file called Procfile. The content should be:

```
web gunicorn manage:app
```

(if this is how you run your application).

In the Procfile, each line has a task name, a colon, and the command that runs the task. We use web here because Heroku recognizes it as the task that start the web server. Heroku gives this task a PORT environment variable, and set it to the port in which the application needs to listen for requests.

Using Foreman to Emulate Heroku

The Heroku Toolbet includes Foreman, used to run the app locally through the Procfile for testing purposes. The environment variables set at Heroku must be defined locally. Just create a file var.env with this information:

```
FLASK_CONFIG=heroku
MAIL_USERNAME=<your-username>
MAIL_PASSWORD=<your-password>
```

Foreman run is used to run commands under the environment of the application. Foreman start reads the Procfile and executes the tasks in it:

```
$ foreman run python manage.py deploy
$ foreman start
```

Configuring Logging

In Heroku, logs are written to stdout or stderr. In my app, I added the logging configuration to a class in my app's config.py file:

```
class HerokuConfig(ProductionConfig):
    @classmethod
    def init_app(cls, app):
        ProductionConfig.init_app(app)

    import logging
```

```
    from logging import StreamHandler
    file_handler = StreamHandler()
    file_handler.setLevel(logging.WARNING)
    app.logger.addHandler(file_handler)
```

To let Heroku know what configuration it should use, I add this environment variable:

```
$ heroku config:set FLASK_CONFIG=heroku
```

Now if something goes wrong when you deploy, you can always check the log:

```
$ heroku logs
```

Deploying!

If everything is good, it's time to deploy your application. Since you already committed your app before, you just need to push it to Heroku:

```
$ git push heroku master
```

In my app, I have a script for the deployment (such taking care of database and other setups for production). So, additionally, I run:

```
$ heroku run python manage.py deploy
$ heroku restart
```

That's it! The app should be running at < app-name >.hero-kuapp.com.

Written by bt3gl

Recommend

Say Thanks

Update Notifications Off

Respond

Filed Under

Ruby on Rails Development Tips

Tools

Sponsors

Awesome Job See All Jobs →

Senior Java Developer
Expeditors · -Seattle, WA · Full Time

Post a job for only $299