CODING NOMADS

19) Deploying Your App On The Internet Lesson

Part 1: Setup for Flask Deployment on Heroku

14 min to complete · By Brandon Gigous

Contents

This is Part 1 of our Flask Deployment tutorial series:

Part 1: Setup for Flask Deployment on Heroku
Part 2: Configuration for Deployment on Heroku
Part 3: Using Git & Debugging Your Deployment on Heroku

After completing this lesson, be sure to complete Parts 2 and 3 to tie it all together!

In this lesson, you will learn - what is Heroku, how Heroku simplifies Flask deployment, and how to set up for deployment on Heroku using Python + Flask.

What is Heroku?

Heroku is a Platform as a Service (PaaS) that has been around since 2007 and provides a stable and user-friendly experience. Heroku has created a powerful and intuitive platform that takes a load of deployment worries off your shoulders, which is ideal when you want to get your project out into the world-wide-world.

Since a PaaS performs work that you would have to do otherwise, these platforms usually charge money for their services. Heroku has an Eco and Basic Tier that allows you to run personal projects for very cheap. That's what you'll be working with in this course section.

Note: Choose the cheapest option available if you'll be following along. You can always upgrade later if you need to.

Before being ready to deploy your app, you will need to revisit a few sections in your code and clean it up for production. Heroku also requires some specific settings that make it possible to launch your app on their platform. Let's get to it so you can finally let your friends and family discover your work through the internet.

Getting Started with Heroku

There are only a few things you need to do to get started with Heroku. Check out their tutorials and:

- Create a free account

- Install the Heroku CLI
- Create your app on Heroku

Install Heroku CLI

After you have created your free account, you can install the Heroku CLI using the following code:

```
# macOS
$ brew install heroku/brew/heroku
# Ubuntu
$ sudo snap install --classic heroku
```

You can then log in to your account through the CLI:

```
$ heroku login
```

It will prompt you to log in through your browser.

One last thing: if you have already been committing your code through `git`, then you can connect your repo to Heroku. This is done with:

```
$ heroku create <appname>
```

The `<appname>` can be anything that isn't already taken, as it must be unique all across Heroku. A couple ideas for you: `flask-webdev-<your initials>`, `ragtime <your name>`

Even if you aren't quite ready to do that last step, there's still plenty of time to learn the basics in the next few lessons. You are already well on your way to deploying your Flask app to Heroku. But first, let's talk about **secrets**.

Manage Secrets in Heroku

Some secrets are meant to stay secret. For your Flask app, first and foremost is your aptly named `SECRET_KEY`. You've already set this in your own app and as you may remember, it's used to protect your app's user sessions and generates your app's tokens. However, you need to make sure that the `SECRET_KEY` you will be using in production is **not visible** in your code base, especially if you are pushing your code to public version control such as GitHub.

Alert: If you accidentally pushed sensitive information to your GitHub account, you need to **assume that it is compromised**, even if you notice it right away. There are bots that constantly scrap public GitHub repositories for such information. That means you will need to **change the information**. If it's your password or API key - change it. There's no way around that if you want to stay safe.

To avoid having to pull your hair and go through stressful damage-control actions, let's rather make sure that your sensitive information stays off the web.

Local Solutions

To avoid committing sensitive information to version control *in general*, you would take it out of the code that you will register in version control. As always, there are different solutions to where to keep this secret information:

- `secrets.py` : A Python file that declares the variables you use that you import from in your main code. That file is excluded from version control via `.gitignore` .
- **Environment Variables**: You register the variables inside of your virtual environment, most conveniently by exporting them directly inside of your `activate` script. If you do this, they will automatically be loaded every time you activate your virtual environment and are accessible in your code via `os.environ` .
- `dotenv` : A similar approach, but as a package. It allows you to specify environment variables as simple key-value pairs and load them into your environment to use. It requires installing a separate package, however, and some extra lines of code.

These solutions work locally to keep your sensitive bits of information from the prying public eye, but what about your deployed application? Let's see what solution Heroku has come up with for you.

Setting Config Vars Through Heroku CLI

Heroku gives you both a **CLI** and a **GUI web interface** to set environment variables for your remote application. You can think of it in the same way as setting your environment variables locally. Below, you will learn the essentials. You can always consult the official documentation on Config Vars.

After you have the Heroku CLI successfully installed and have created your Heroku app, you can quickly set config vars in your remote environment similarly to how you would do that locally using `export` replacing `<YOUR_SECRET_KEY>` with your own secret key:

```
$ heroku config:set SECRET_KEY=<YOUR_SECRET_KEY>
```

With similar commands, you can also read (`get`) and remove (`unset`) config vars.

Remember to make the secret key long and hard to crack! It's no good having a secret key as `secret` or `password` . Any easy to find password cracking software will crack that in no time.

For your secret key in production, it's recommended to use something like Python's `uuid` module to generate a random string in hex from your terminal like so:

```
$ python -c "import uuid; print(uuid.uuid4().hex)"
995366f815394c0eab0e2f8bbd50f1cb
```

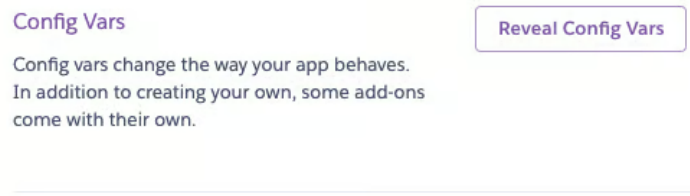
Then, you can use the `heroku config` command like shown above to use the output as your app's `SECRET_KEY` when deployed through Heroku.

Setting Config Vars Through Heroku's Dashboard

Alternatively, you can use the web GUI interface to view, add, and remove config variables for your remote environment on a per-app basis.

Access your Heroku Dashboard, click to access your app, then click on the "Settings" tab.

Here you have the option to click "Reveal Config Vars". Be brave and click that button:



You will be presented with a GUI view of your current config vars, which is also an interface to set, edit, and delete them. This is where you'll want to add your `SECRET_KEY`:

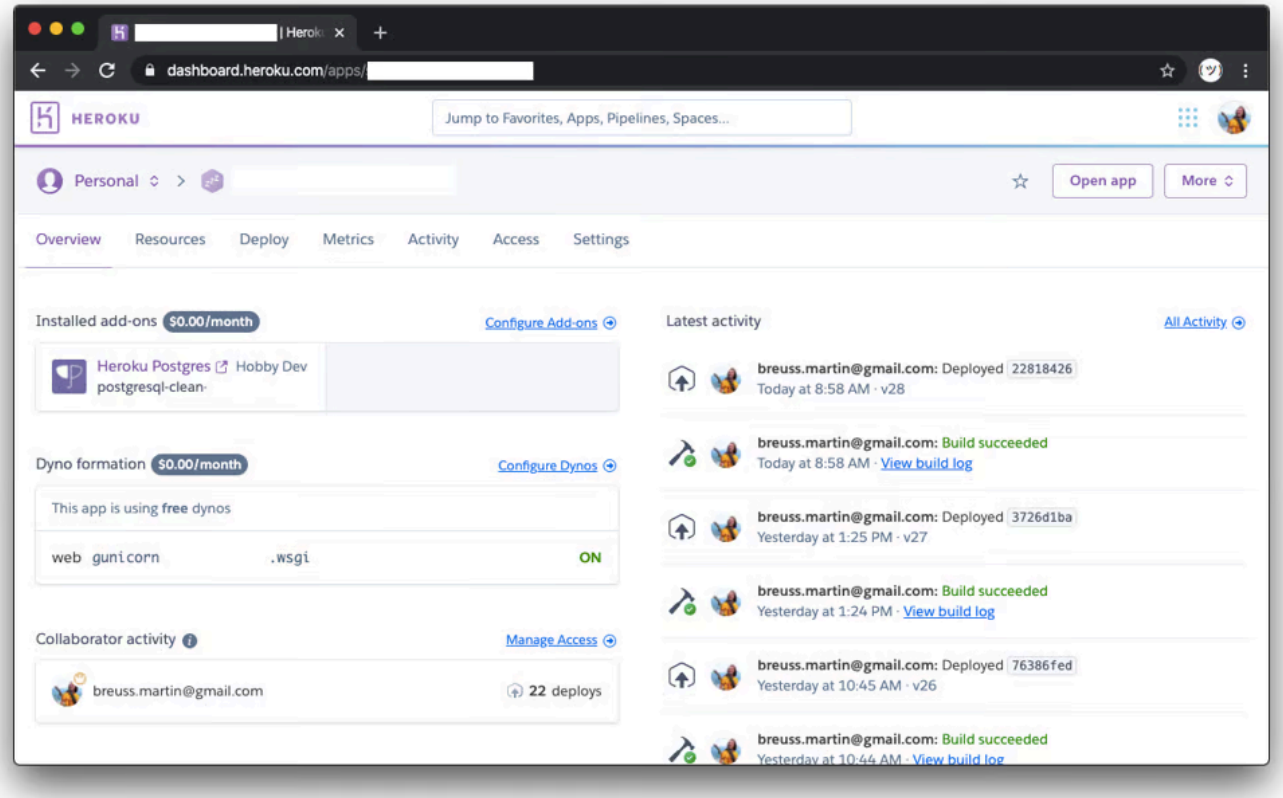
Other Variables To Set

Once you have the `SECRET_KEY` set, you'll want to set these other variables on the Heroku side as well:

- `FLASK_APP`: Set to `ragtime.py`, of course!
- `FLASK_DEBUG`: Set to `0` since you don't want to reveal any debug information to users
- `FLASK_CONFIG`: Set to `heroku`; you'll see more about this value later
- `MAIL_USERNAME`: Set to `<your email username>`
- `MAIL_PASSWORD`: Set to `<your email password>`; this will be the *app password* for Gmail!

Why Use the Heroku Dashboard

As you've seen in the previous exercise, Heroku provides a convenient Graphical User Interface that allows you to set, control, and monitor many aspects of your deployed application.



This is one of the features of using a PaaS: Heroku picks and decides how to handle your deployment for you and offers you pre-defined ways to interact with it. The GUI is another feature that makes these interactions more convenient.

Go ahead and access your Heroku Dashboard if you haven't done so in the previous step, and take some time to explore what it has to offer.

Then check out Heroku's official learning resource about their dashboard.

Secrets Saved!

Adding your `SECRET_KEY` variable to the config vars, either through the Dashboard or the Heroku CLI, is an important step to getting your Flask app deployed properly. However, there are a few changes you will need to make in order to avoid breaking your app when doing so. Let's look at those in Part 2 of the Flask deployment on Heroku tutorial series!

Summary: Setup for Flask Deployment on Heroku

In this lesson, you:

- Got acquainted with **Heroku**, a Platform as a Service (PaaS) that simplifies app deployment.
- Installed the **Heroku CLI** and learned to log in via the terminal.

Previous Next → Part 2: Configuration for Deployment on Heroku