Part 3: Using Git & Debugging Your Deployment on Heroku

11 min to complete · By Brandon Gigous

Contents

- Introduction
- Integrating Heroku with Git
  - Git Version Control
  - Create A Heroku App
  - Add The `heroku` Remote
  - Push Your Code
- Run Deploy Tasks
- Debugging Deployment
  - Check the logs
  - Try running locally
  - Check (un)committed files
- Summary: Flask Deployment on Heroku

Welcome to Part 3 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

If you haven't already completed Parts 1 and 2, please visit those lessons and return here!

In this lesson, you will learn how to use Git for deployment on Heroku, create a Heroku app, and debug your Flask deployment on Heroku.

Integrating Heroku with Git

Heroku has a seamless integration with Git, which gives you an easy way to deploy your app that works well with a normal development workflow. There are four things you need to do:

- Put your project code under version control with Git
- Create a Heroku app
- Add a remote named `heroku` to your repository
- Push your code to the `heroku` remote

Let's look at each of these steps one after another. You may not need to do some of them anymore, depending on how you set up your project so far. You can read more in Heroku's guide on Deploying With Git.

Git Version Control

Assuming you've been closely following along, you already have your project under version control with git. But if you haven't yet put your project under version control, *now* is the time to do so. Go to the root folder of your project in your terminal, then type:

```
$ git init
```

Add and commit everything, *except* any files containing sensitive information. Your project is now under version control. Yay! If you need a Git refresher, check out the link in the Prerequisites section at the beginning of this course.

Create A Heroku App

You might have already done this, if you're following along, but if not, you can either create a Heroku app from your Heroku Dashboard, or from the Heroku CLI. In this step, you can give your app a **name**.

Info: The name you give to your Heroku app in this step will be part of the URL that your webapp will be accessible at: `https://your-app-name.herokuapp.com` Choose something you can remember, but keep in mind that the name must not be in use yet. Heroku will inform you if the app name is not available.

The easiest is to type the following command again from the root project folder that you moved into when initiating your Git repository:

```
$ heroku create your-app-name
```

Replace `your-app-name` with the name of the app you want to use. It needs to be available for the app to get created successfully.

Add The `heroku` Remote

If you created the app via the Heroku CLI as described above, you can skip this step, as Heroku already completed it for you. To check, run the command `git remote -v`, and you should see a remote called heroku:

```
heroku  https://git.heroku.com/<your-app-name>.git (fetch)
heroku  https://git.heroku.com/<your-app-name>.git (push)
```

Check the docs for how to link a `heroku` remote of your Git repository to a Heroku app you created through the Dashboard instead.

Push Your Code

The final step is pushing your code to the `heroku` remote in the same way you would push to the `origin` remote on GitHub:

```
$ git push heroku master
```

This will **push and deploy** your webapp on Heroku under the `https://your-app-name.herokuapp.com` . You can watch the deployment process take place in your command line. Once it is finished, the CLI will also show you the URL to which your web app has just been deployed. *But hold your Heroku horses!*

Run Deploy Tasks

By this point, your app is *technically* deployed; however, you need to initiate the `deploy` command that you previously created. To do that, you use the `heroku run` command:

```
$ heroku run flask deploy
```

Then, to make sure you have a clean start with a newly created/upgraded database, you'll want to restart your deployed app:

```
$ heroku restart
```

Okay, okay, yes, *now* you can try that link to your app, or copy-paste it into your browser, and you're finally ready to see your webapp live on the internet! You can share this link with anyone on the web, and they will now be able to access your page. Well done, your app is open for the world to see! :D

Debugging Deployment

Is your app showing an error, acting weirdly, or just plain not showing up? Here are a few ways you can go about debugging your deployment:

Check the logs

Heroku allows you to see your deployed app's log messages both directly from the terminal or through the Heroku Dashboard on their website. To check the logs, use:

```
# optional: use -t to continually receive incoming messages
$ heroku logs [-t]
```

Try running locally

The Heroku CLI comes with a neat feature: `heroku local` allows you to "dry run" your deployment by running the application locally in a similar way to how Heroku would run your app on its own servers. You'll need to set up the environment variables yourself, however, as Heroku won't grab them for you. This command will look for a top-level `.env` file that contains your environment variables and their values:

```
FLASK_APP=ragtime.py
FLASK_CONFIG=heroku
MAIL_USERNAME=<your-gmail-username>
MAIL_PASSWORD=<your-gmail-password>
```

Note: Remember not to commit this file to version control! You can add a line to your `.gitignore` to ignore the `.env` file.

But just like setting your app up on Heroku, you'll need to run the `deploy` command. Afterwards, you can then run it locally:

```
$ heroku local:run flask deploy
$ heroku local
```

Check (un)committed files

Perhaps you missed something and didn't push that *one* file needed for your app to work. Could it be you didn't commit your entire `migrations` folder? Or maybe you forgot that latest change to `app/__init__.py`.

If you've tried every feature of your Internet-facing app and it all works, great job! You've successfully deployed your app.

Summary: Flask Deployment on Heroku

In this three-part Flask Deployment on Heroku tutorial series, 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.
- Discovered how to manage **environment variables** on Heroku using `heroku config:set`, or with the GUI.
- Tackled the application security by setting up **TLS with Flask-Talisman** and enabled CSRF protection on non-form endpoints with **Flask-WTF CSRFProtect**.
- Addressed **deployment tasks**, such as database migrations and role creations, using custom Flask CLI commands.
- Wrote a `Procfile` to declare how Heroku should run your app.
- Wrote a `requirements.txt` for tracking dependencies to be installed by Heroku during deployment.
- Realized how to **use Git for deployment** on Heroku, setting up a remote named 'heroku' and pushing your code to deploy the app.