



# Deploying a Flask App

Learn how to deploy your own Flask application with Heroku.

One of the best parts about building cool websites, is being able to share them with your friends, family and future employers.

Up until this point however, we've been running our website on localhost, in a development environment.

While this is great for making changes and developing features quickly, it's not ideal for when grandma asks to see what you've been building (unless grandma knows your IP address that is).

*Enter deployment.*

## Deployment

Deployment is the process of taking the amazing local website you've been working on, and packaging it up so that other people can use it. In the case of web development, this generally means hosting the site on a publicly accessible web server.

There are a number of places you can deploy and host a web app, including a web server that you build yourself. But most developers utilize what's called a platform-as-a-service (PAAS).

A PAAS is an online platform which is designed to be able to easily run, scale and

**Next**

**Get Help**

you can upload your project's source code to one of these services, and it will handle all the configuration, version control and maintenance tasks necessary to run it.

## Heroku

By far the most popular and beginner friendly PAAS is Heroku.

Heroku is the recommended option for beginners because it's free for small projects, widely used in the industry, and makes deploying a flask app a piece of cake.

In the next couple exercises, we'll look at how to get Heroku's command line tools installed on your computer, and how easy it is to deploy your app.

## Heroku CLI

When we think about deployment, we need to think in terms of two environments:

- **Development environment** - Your local development server where you run your app on localhost and make changes / develop new features
- **Production environment** - The publicly accessible web server where the finished version of your app is running (in this case, Heroku's Platform as a Service).

At this point you're familiar with the development environment. This is where you'll spend most of your time as a web developer.

But once you're done developing, you need a way of uploading and running your code on the Heroku production environment.

## Heroku Command Line Interface (CLI)

The Heroku CLI is a software application that runs in the command line and is able to assist you in uploading the code in your development environment to the Heroku production environment.

Next

Get Help

For the CLI to work, you first need to create a free account on the Heroku website. Head over to [heroku.com](https://heroku.com), click the “Sign up” button, and enter in your relevant details.

Once you have an account, it's time to install the command line tools.

## Install the CLI Tools

In the workspace on the right you'll find a detailed instructions page for installing the CLI on your operating system (Windows, Linux or OSX).

Do your best to follow along, it shouldn't take more than a couple minutes.

Once that's installed, verify it's working by running the following in your command line:

```
heroku --version
```

Next, you'll need to login to your newly created heroku account from inside the command line, so Heroku knows where to upload your app to.

Run the following command, and follow the prompts:

```
heroku login
```

At this point, you should have the CLI installed and be logged in and ready to deploy.

## Create a Heroku/Git Project

In this section, we'll transform your project's directory into a git repository, and create a new heroku app for it which will be linked to our local project.

**Next**

**Get Help**

*GIT is a version control system that makes it simple to track changes and bookmark development milestones in your projects, we'll eventually use git to deploy our app to heroku. (For the uninitiated, check out Codecademy's git course [git course](#))*

In the same terminal window, run the following:

```
git init
heroku create
```

```
Creating app... done, 🍄 sleepy-meadow-81798
https://sleepy-meadow-81798.herokuapp.com/ | https://git.heroku.com/sleepy-meadow-817
```

The output should look something like the above.

Heroku will automatically create a custom domain name for your new app (in the above case <https://sleepy-meadow-81798.herokuapp.com/>). This is where you can access your app once you deploy it.

## Swapping SQLite for PostgreSQL

Up until this point, we've been using SQLite as our database, which is good because it's simple and easy to setup.

When we deploy our app to Heroku however, we'll want to use a database that is better suited for production, and able to scale properly with our app.

## PostgreSQL

PostgreSQL is an open source Relational Database Management System (RDBMS), which is used in some of the most popular apps in the world.

Our flask project can communicate with it just like it communicates with SQLite, so we'll only need to change one line of code to make it work.

We'll set things up so our app only uses PostgreSQL when it's running in production

[Next](#)

[Get Help](#)

Heroku will handle all of the PostgreSQL configurations and setup, we just need to let it know we want to use it.

```
heroku addons:create heroku-postgresql:hobby-dev
```

Start by running the above command in your project terminal, which adds the `postgresql` addon to our heroku app (`hobby-dev` is the free PostgreSQL tier).

Next we need to install a library which will allow SQLAlchemy (the library we're using to communicate with our database) to talk to PostgreSQL.

```
pip install psycopg2
```

Finally we'll modify our code so that the app uses SQLite when we're developing and PostgreSQL when in production.

If you've been following along with the module up to this point you should have a line of code that looks something like this:

```
app.config['SQLALCHEMY_DATABASE_URI'] = "sqlite:///myDB.db"
```

This line tells SQLAlchemy where your database is. Let's modify the code so that it instead looks like the following:

```
from os import environ # this line should go at the top of your file
...
app.config['SQLALCHEMY_DATABASE_URI'] = environ.get('DATABASE_URL') or
'sqlite:///myDB.db'
```

In the modified code above, we're using an [environment variable](#) called `DATABASE_URL` to tell SQLAlchemy where our database is located.

Next

Get Help

When we added PostgreSQL to our heroku project, it automatically created that `DATABASE_URL` environment variable for us. So when our code is run on Heroku, `os.environ['DATABASE_URL']` should automatically point to the PostgreSQL database.

## Deployment Prep

It's almost time to deploy!

But wait... Before we do, let's make sure your project's files and dependencies are in order.

For Heroku to accept our app for deployment we need to add two files to the root of our project, and install an additional dependency:

- **requirements.txt** - a file that specifies all the dependencies your app relies on
- **Procfile** - a file which tells Heroku how to run our app (This file has no extension)
- **Gunicorn** - a web server dependency which Heroku will use to serve our app in production.

Let's start with Gunicorn, which you can install using pip with the following command:

```
pip install gunicorn
```

The web server that flask uses for local development isn't powerful enough for use in production, so gunicorn to the rescue!

Next, to create the requirements.txt file, simply run the command:

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

And pip will automatically figure out your app's dependencies, and chuck them into

Next

Get Help

case we'll tell it to use the gunicorn web server instead of the development server we use on our local machine.

Place the following text in a file named `Procfile` at the root directory of your project:

```
web: gunicorn <module-name>:<app-name>
```

Replace `<module-name>` with the name of the module or file that holds your main flask controller file, and `<app-name>` with the name of your flask app.

In most cases, and if you've been following along with this course, the module name is `app` because the flask code is in the file `app.py`, and the app name is `app`, because that's what we called it in the file.

Remember, these files go in the root directory of our project.

Once the `requirements.txt` and `Procfile` are squared away, it's finally time to deploy!

## Deploying the App

It's finally time to deploy!

While there are several ways you can deploy your app up to Heroku, the easiest is to use git.

## Setting up Git

Your project's directory should already be initialized as a git repository, the only thing left to do is create a commit of all your code.

Open your terminal to the base directory of your flask project and run the following commands:

**Next**

**Get Help**

This will stage and commit all your code (indicating to Heroku that it's ready to be deployed).

## Push your Site

Finally, push your code up to the remote production environment by running:

```
git push heroku master
```

This may take a few seconds, but when you're done your app should be running in production!

If you have any SQLAlchemy models in your project you'll need to create them on the new PostgreSQL database. Run the following commands:

```
heroku python run
```

```
>>> from app import db
>>> db.create_all()
```

The above will open an interactive python terminal on your heroku app, and allow you to create all the database models remotely.

Test that everything works by running:

```
heroku open
```

## App Maintenance

Once your app is deployed, the only thing left to do is make sure it stays that way.

While maintaining an app on Heroku is not rocket science, it's important to at least have some awareness of how you might go about making sure it's running smoothly.

Most interesting tasks will take place from the Heroku CLI, but you can also use the Heroku Dashboard.

[Next](#)

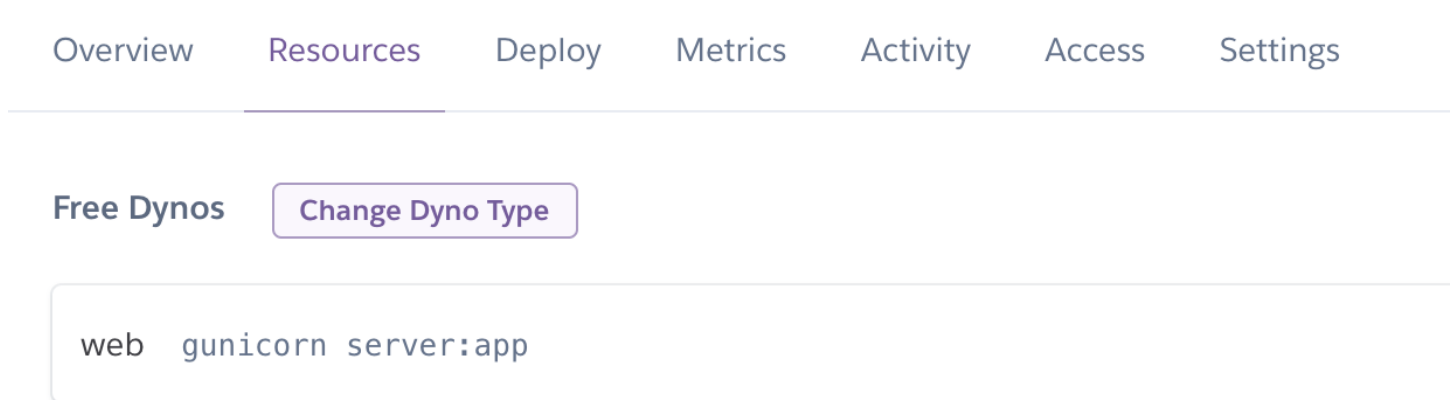
[Get Help](#)



Heroku runs your app on something called a Dyno. Without getting overly technical, it's essentially a computing container that is solely dedicated to running your app.

Each new project on Heroku gets a free Dyno with 512mb of RAM (memory), and an unspecified amount of processing power.

When you're first starting out, this is the perfect amount and price (free). But as your app grows, and requires more and more resources to run, you can manually increase the number and power of Dynos dedicated to your app in the Heroku web dashboard.

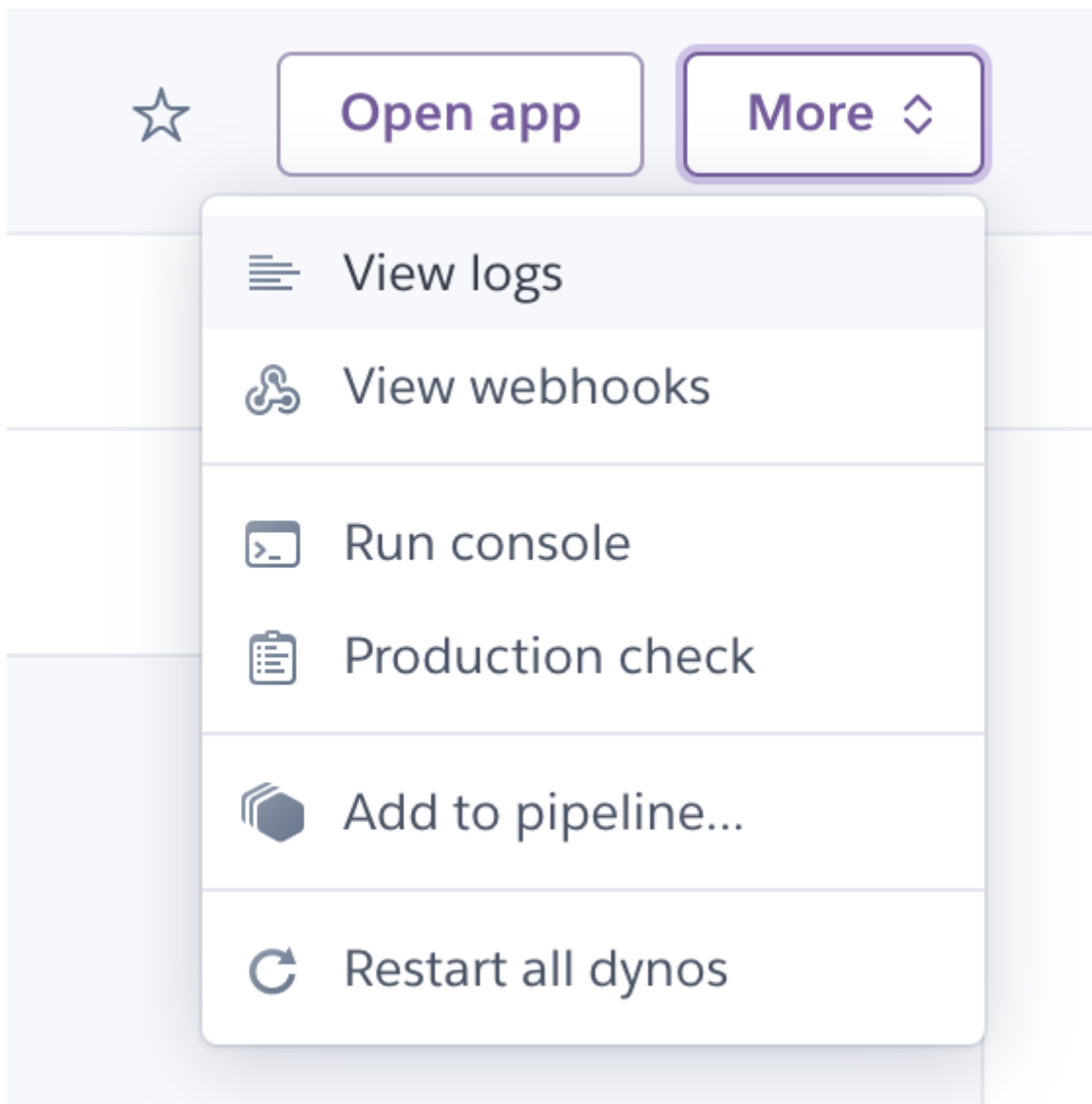


In the resources tab, select “Change Dyno Type”, and Heroku will show you the paid plans you can upgrade to.

## Logs

When an app is in production, it's useful to be able to check on how it's doing from time to time, and take note of any errors or problems that it encounters.

To help with this, the Heroku web dashboard has a section under the “more” button on the top right of the toolbar, where you can view your application's output logs.



Anything that your app outputs to the console will show up here, including custom messages you added in yourself.

## Continuous Deployment

Continuous Deployment is a process whereby changes made to your web app are automatically deployed to Heroku without you having to do anything.

In the previous exercise we saw how Heroku apps can be deployed using git. It's possible to take this a step further however by linking your Heroku app up with GitHub.

**Next**

**Get Help**

If you are already storing your project on GitHub, you can tell Heroku to watch for changes made to the master branch of your GitHub repository, and automatically deploy them.

This means that anytime you push or merge a change into your master branch on GitHub, Heroku will deploy it automatically.



In the “Deploy” tab on the main web dashboard interface, select the “Connect to Github” option, and follow the prompts that appear.

[Deploy](#) [Metrics](#) [Activity](#) [Access](#) [Settings](#)




---

Use an existing pipeline or create a new one.


Add this app to a stage in a pipeline to enable additional features




Pipelines let you connect multiple apps together and **promote code** between them. [Learn more.](#)





Pipelines connect your apps, and you can promote code between them. [Learn more.](#)



Choose a pipeline

 **Heroku Git**  
Use Heroku CLI

 **GitHub**  
Connect to GitHub

 **Container Registry**  
Use Heroku CLI

This should be a pretty straightforward integration, and once done, you won't have to worry about deploying to Heroku anymore!

[Next](#)[Get Help](#)