

# Getting Started on Heroku with Python

🕒 Last updated 03 September 2020

## ☰ Table of Contents

- Introduction
- Set up
- Prepare the app
- Deploy the app
- View logs
- Define a Procfile
- Scale the app
- Declare app dependencies
- Run the app locally
- Push local changes
- Provision add-ons
- Start a console
- Define config vars
- Provision a database
- Next steps

## Introduction

This tutorial will have you deploying a Python app (a simple Django app) in minutes.

Hang on for a few more minutes to learn how it all works, so you can make the most out of Heroku.

The tutorial assumes that you have:

- a free Heroku account (<https://signup.heroku.com/signup/dc>).
- Python version 3.7 installed locally - see the installation guides for OS X (<http://docs.python-guide.org/en/latest/starting/install3/osx/>), Windows (<http://docs.python-guide.org/en/latest/starting/install3/win/>), and Linux (<http://docs.python-guide.org/en/latest/starting/install3/linux/>).
- Postgres installed (<https://devcenter.heroku.com/articles/heroku-postgresql#local-setup>) locally, if running the app locally.

## Set up



The Heroku CLI requires **Git**, the popular version control system. If you don't already have Git installed, complete the following before proceeding:

- Git installation (<https://git-scm.com/book/en/v2/Getting-Started-Installing-Git>)
- First-time Git setup (<https://git-scm.com/book/en/v2/Getting-Started-First-Time-Git-Setup>)

In this step you'll install the Heroku Command Line Interface (CLI). You use the CLI to manage and scale your applications, provision add-ons, view your application logs, and run your application locally.

Download and run the installer for your platform:



**Download the installer (<https://cli-assets.heroku.com/heroku.pkg>)**

Also available via Homebrew:

```
$ brew install heroku/brew/heroku
```



Download the appropriate installer for your Windows installation:

**64-bit installer (<https://cli-assets.heroku.com/heroku-x64.exe>)**

**32-bit installer (<https://cli-assets.heroku.com/heroku-x86.exe>)**



Run the following from your terminal:

```
$ sudo snap install heroku --classic
```

Once installed, you can use the `heroku` command from your command shell.

Use the `heroku login` command to log in to the Heroku CLI:

```
$ heroku login
heroku: Press any key to open up the browser to login or q to exit
> Warning: If browser does not open, visit
> https://cli-auth.heroku.com/auth/browser/**
heroku: Waiting for login...
Logging in... done
Logged in as me@example.com
```

This command opens your web browser to the Heroku login page. If your browser is already logged in to Heroku, simply click the **Log in** button displayed on the page.

This authentication is required for both the `heroku` and `git` commands to work correctly.



If you're behind a firewall that requires use of a proxy to connect with external HTTP/HTTPS services, **you can set the `HTTP_PROXY` or `HTTPS_PROXY` environment variables** (<https://devcenter.heroku.com/articles/using-the-cli#using-an-http-proxy>) in your local development environment before running the `heroku` command.

## Prepare the app

In this step, you will prepare a simple application that can be deployed.

To clone the sample application so that you have a local version of the code that you can then deploy to Heroku, execute the following commands in your local command shell or terminal:

```
$ git clone https://github.com/heroku/python-getting-started.git
$ cd python-getting-started
```

You now have a functioning git repository that contains a simple application, a `runtime.txt` specifying Python 3.7.6, and a `requirements.txt`, which is used by Python's dependency manager, Pip.

## Deploy the app

In this step you will deploy the app to Heroku.

Create an app on Heroku, which prepares Heroku to receive your source code:

```
$ heroku create
Creating app... done, ● serene-caverns-82714
https://serene-caverns-82714.herokuapp.com/ | https://git.heroku.com/serene-caverns-82714.git
```

When you create an app, a git remote (called `heroku`) is also created and associated with your local git repository.

Heroku generates a random name (in this case `serene-caverns-82714`) for your app, or you can pass a parameter to specify your own app name.

Now deploy your code:

```

$ git push heroku main
Counting objects: 407, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (182/182), done.
Writing objects: 100% (407/407), 68.65 KiB | 68.65 MiB/s, done.
Total 407 (delta 199), reused 407 (delta 199)
remote: Compressing source files... done.
remote: Building source:
remote:
remote: -----> Python app detected
remote: -----> Installing python-3.7.6
remote: -----> Installing pip
remote: -----> Installing SQLite3
remote: -----> Installing requirements with pip
remote: Collecting django (from -r /tmp/build_394859b69f6aeb1b63e599ce5b6c69bd/requirements.txt (line 1))
remote:   Downloading https://files.pythonhosted.org/packages/32/ab/22530cc1b2114e6067eece94a333d6c749fa1c56a009f0721e51c181ea53/Django-2.1.2
remote: Collecting gunicorn (from -r /tmp/build_394859b69f6aeb1b63e599ce5b6c69bd/requirements.txt (line 2))
remote:   Downloading https://files.pythonhosted.org/packages/8c/da/b8dd8deb741bff556db53902d4706774c8e1e67265f69528c14c003644e6/gunicorn-19.
remote: Collecting django-heroku (from -r /tmp/build_394859b69f6aeb1b63e599ce5b6c69bd/requirements.txt (line 3))
remote:   Downloading https://files.pythonhosted.org/packages/59/af/5475a876c5add5a3494db47d9f7be93cc14d3a7603542b194572791b6c6/django_herok
remote: Collecting pytz (from django->-r /tmp/build_394859b69f6aeb1b63e599ce5b6c69bd/requirements.txt (line 1))
remote:   Downloading https://files.pythonhosted.org/packages/30/4e/27c34b62430286c6d59177a0842ed90dc789ce5d1ed740887653b898779a/pytz-2018.5-
remote: Collecting psycpg2 (from django-heroku->-r /tmp/build_394859b69f6aeb1b63e599ce5b6c69bd/requirements.txt (line 3))
remote:   Downloading https://files.pythonhosted.org/packages/37/88/40748331bf75d068a07bbae7dc658faceb0ce2e9ffffdde550e76d5475e59/psycpg2-2.7
remote: Collecting dj-database-url>=0.5.0 (from django-heroku->-r /tmp/build_394859b69f6aeb1b63e599ce5b6c69bd/requirements.txt (line 3))
remote:   Downloading https://files.pythonhosted.org/packages/d4/a6/4b8578c1848690d0c307c7c0596af2077536c9ef2a04d42b00fabaa7e49d/dj_database_
remote: Collecting whitenoise (from django-heroku->-r /tmp/build_394859b69f6aeb1b63e599ce5b6c69bd/requirements.txt (line 3))
remote:   Downloading https://files.pythonhosted.org/packages/07/2e/c77e71cb448f1a507bc2dfec1d5c24e35d14a737837bea6cdfd6d1ff66bd/whitenoise-4
remote: Installing collected packages: pytz, django, gunicorn, psycpg2, dj-database-url, whitenoise, django-heroku
remote: Successfully installed dj-database-url-0.5.0 django-2.1.2 django-heroku-0.3.1 gunicorn-19.9.0 psycpg2-2.7.5 pytz-2018.5 whitenoise-4
remote: -----> $ python manage.py collectstatic --noinput
remote:   131 static files copied to '/tmp/build_a72377875f1c522b087e93a543bac7d5/staticfiles', 411 post-processed.
remote:
remote: -----> Discovering process types
remote:   Procfile declares types -> web
remote:
remote: -----> Compressing...
remote:   Done: 57.1M
remote: -----> Launching...
remote:   Released v5
remote:   https://serene-caverns-82714.herokuapp.com/ deployed to Heroku
remote:
remote: Verifying deploy... done.
To https://git.heroku.com/serene-caverns-82714.git
 * [new branch]      revert-to-requirements -> main

```

The application is now deployed. Ensure that at least one instance of the app is running:

```
$ heroku ps:scale web=1
```

Now visit the app at the URL generated by its app name. As a handy shortcut, you can open the website as follows:

```
$ heroku open
```

## View logs

Heroku treats logs as streams of time-ordered events aggregated from the output streams of all your app and Heroku components, providing a single channel for all of the events.

View information about your running app using one of the logging commands (<https://devcenter.heroku.com/articles/logging>), `heroku logs --tail`:

```

$ heroku logs --tail
2018-10-12T19:13:57.748721+00:00 heroku[web.1]: Starting process with command `gunicorn gettingstarted.wsgi`
2018-10-12T19:13:59.308299+00:00 app[web.1]: [2018-10-12 19:13:59 +0000] [4] [INFO] Starting gunicorn 19.9.0
2018-10-12T19:13:59.308880+00:00 app[web.1]: [2018-10-12 19:13:59 +0000] [4] [INFO] Using worker: sync
2018-10-12T19:13:59.308777+00:00 app[web.1]: [2018-10-12 19:13:59 +0000] [4] [INFO] Listening at: http://0.0.0.0:3142 (4)
2018-10-12T19:13:59.313176+00:00 app[web.1]: [2018-10-12 19:13:59 +0000] [10] [INFO] Booting worker with pid: 10
2018-10-12T19:13:59.331441+00:00 app[web.1]: [2018-10-12 19:13:59 +0000] [11] [INFO] Booting worker with pid: 11
2018-10-12T19:13:59.864677+00:00 heroku[web.1]: State changed from starting to up
2018-10-12T19:14:03.000000+00:00 app[api]: Build succeeded
2018-10-12T19:19:00.370216+00:00 heroku[router]: at=info method=GET path="/" host=serene-caverns-82714.herokuapp.com request_id=308ae087-635f-4cf8-8

```

Visit your application in the browser again, and you'll see another log message generated.

Press `Control+C` to stop streaming the logs.

## Define a Procfile

Use a Procfile (<https://devcenter.heroku.com/articles/procfile>), a text file in the root directory of your application, to explicitly declare what command should be executed to start your app.

The Procfile in the example app you deployed looks like this:

```
web: gunicorn gettingstarted.wsgi --log-file -
```

This declares a single process type, `web`, and the command needed to run it. The name `web` is important here. It declares that this process type will be attached to the HTTP routing (<https://devcenter.heroku.com/articles/http-routing>) stack of Heroku, and receive web traffic when deployed.

Procfiles can contain additional process types. For example, you might declare one for a background worker process that processes items off of a queue.

## Microsoft Windows

The sample app has an additional Procfile for local development on Microsoft Windows, located in the file `Procfile.windows`. Later tutorial steps will use this instead: it starts a different web server, one that is compatible with Windows.

```
web: python manage.py runserver 0.0.0.0:5000
```

## Scale the app

Right now, your app is running on a single web dyno (<https://devcenter.heroku.com/articles/dynos>). Think of a dyno as a lightweight container that runs the command specified in the Procfile.

You can check how many dynos are running using the `ps` command:

```
$ heroku ps
Free dyno hours quota remaining this month: 999h 6m (99%)
Free dyno usage for this app: 0h 0m (0%)
For more information on dyno sleeping and how to upgrade, see:
https://devcenter.heroku.com/articles/dyno-sleeping

=== web (Free): gunicorn gettingstarted.wsgi (1)
web.1: up 2018/10/12 14:26:45 -0500 (~ 33s ago)
```

By default, your app is deployed on a free dyno. Free dynos will sleep after a half hour of inactivity (if they don't receive any traffic). This causes a delay of a few seconds for the first request upon waking. Subsequent requests will perform normally. Free dynos also consume from a monthly, account-level quota of free dyno hours (<https://devcenter.heroku.com/articles/free-dyno-hours>) - as long as the quota is not exhausted, all free apps can continue to run.

To avoid dyno sleeping, you can upgrade to a hobby or professional dyno type as described in the Dyno Types (<https://devcenter.heroku.com/articles/dyno-types>) article. For example, if you migrate your app to a professional dyno, you can easily scale it by running a command telling Heroku to execute a specific number of dynos, each running your web process type.

Scaling an application on Heroku is equivalent to changing the number of dynos that are running. Scale the number of web dynos to zero:

```
$ heroku ps:scale web=0
```

Access the app again by hitting refresh on the web tab, or `heroku open` to open it in a web tab. You will get an error message because you no longer have any web dynos available to serve requests.

Scale it up again:

```
$ heroku ps:scale web=1
```

For abuse prevention, scaling a non-free application to more than one dyno requires account verification (<https://devcenter.heroku.com/articles/account-verification>).

## Declare app dependencies

Heroku recognizes an app as a Python app by looking for key files. Including a `requirements.txt` in the root directory is one way for Heroku to recognize your Python app.

The demo app you deployed already has a `requirements.txt`, and it looks something like this:

```
django
gunicorn
django-heroku
```

The `requirements.txt` file lists the app dependencies together. When an app is deployed, Heroku reads this file and installs the appropriate Python dependencies using the `pip install -r` command.

To do this locally, you can run the following command:

```
$ pip install -r requirements.txt
```

**Note:** Postgres must be properly installed (<https://devcenter.heroku.com/articles/heroku-postgresql#local-setup>) in order for this step to work properly.

Note: if you're running Linux, the `libpq-dev` system package (or equivalent for your distribution) must also be installed.

Installing the dependencies also caused several other dependencies to be installed. You can see them by using pip's feature `list`:

```
$ pip list
Package            Version
-----
dj-database-url    0.5.0
Django             2.1.2
django-heroku      0.3.1
gunicorn           19.9.0
pip               10.0.1
psycpg2            2.7.5
pytz               2018.5
setuptools         39.0.1
whitenoise         4.1
```

Once dependencies are installed, you will be ready to run your app locally.

## Run the app locally

The app is almost ready to start locally. Django uses local assets, so first, you'll need to run `collectstatic` :

```
$ python manage.py collectstatic
```

Respond with "yes".

Now start your application locally using `heroku local` , which was installed as part of the Heroku CLI.

If you're on Microsoft Windows system, run this:

```
$ heroku local web -f Procfile.windows
```

If you're on a Unix system, just use the default `Procfile` by running:

```
$ heroku local web
```

Your local web server will then start up:

```
[OKAY] Loaded ENV .env File as KEY=VALUE Format
2:28:11 PM web.1 | [2018-10-12 14:28:11 -0500] [18712] [INFO] Starting gunicorn 19.9.0
2:28:11 PM web.1 | [2018-10-12 14:28:11 -0500] [18712] [INFO] Listening at: http://0.0.0.0:5000 (18712)
2:28:11 PM web.1 | [2018-10-12 14:28:11 -0500] [18712] [INFO] Using worker: sync
2:28:11 PM web.1 | [2018-10-12 14:28:11 -0500] [18715] [INFO] Booting worker with pid: 18715
```

Just like Heroku, `heroku local` examines the `Procfile` to determine what to run.

Open `http://localhost:5000` (`http://localhost:5000`) with your web browser. You should see your app running locally.

To stop the app from running locally, go back to your terminal window and press `Ctrl + C` to exit.

## Push local changes

In this step you'll learn how to propagate a local change to the application through to Heroku. As an example, you'll modify the application to add an additional dependency and the code to use it.

Install `requests` locally:

```
$ pip install requests
```

And then add it to your `requirements.txt` file:

```
django
gunicorn
django-heroku
requests
```

Modify `hello/views.py` so that it imports the `requests` module at the start:

```
import requests
```

Now modify the `index` method to make use of the module. Try replacing the current `index` method with the following code:

```
def index(request):
    r = requests.get('http://httpbin.org/status/418')
    print(r.text)
    return HttpResponse('<pre>' + r.text + '</pre>')
```

Now test locally:

```
$ heroku local
```

Visit your application at `http://localhost:5000` (`http://localhost:5000`). You should now see the output of fetching `http://httpbin.org/status/418`, which is a lovely teapot:

```
--[ teapot ]--
```



Now deploy. Almost every deploy to Heroku follows this same pattern. First, add the modified files to the local git repository:

```
$ git add .
```

Now commit the changes to the repository:

```
$ git commit -m "Demo"
```

Now deploy, just as you did previously:

```
$ git push heroku main
```

Finally, check that everything is working:

```
$ heroku open
```

## Provision add-ons

Add-ons are third-party cloud services that provide out-of-the-box additional services for your application, from persistence through logging to monitoring and more.

By default, Heroku stores 1500 lines of logs from your application. However, it makes the full log stream available as a service - and several add-on providers have written logging services that provide things such as log persistence, search, and email and SMS alerts.

In this step you will provision one of these logging add-ons, Papertrail.

Provision the papertrail (<https://devcenter.heroku.com/articles/papertrail>) logging add-on:

```
$ heroku addons:create papertrail
Adding papertrail on sharp-rain-871... done, v4 (free)
Welcome to Papertrail. Questions and ideas are welcome (support@papertrailapp.com). Happy logging!
Use `heroku addons:docs papertrail` to view documentation.
```

To help with abuse prevention, provisioning an add-on requires account verification (<https://devcenter.heroku.com/articles/account-verification>). If your account has not been verified, you will be directed to visit the verification site (<https://heroku.com/verify>).

The add-on is now deployed and configured for your application. You can list add-ons for your app like so:

```
$ heroku addons
```

To see this particular add-on in action, visit your application's Heroku URL a few times. Each visit will generate more log messages, which should now get routed to the papertrail add-on. Visit the papertrail console to see the log messages:

```
$ heroku addons:open papertrail
```

Your browser will open up a Papertrail web console, showing the latest log events. The interface lets you search and set up alerts:

```
May 13 14:43:03 jonwashere heroku/web.1: State changed from down to starting
May 13 14:43:05 jonwashere heroku/web.1: Starting process with command `node web.js`
May 13 14:43:07 jonwashere app/web.1: Listening on 26766
May 13 14:43:08 jonwashere heroku/web.1: State changed from starting to up
May 13 14:43:09 jonwashere heroku/router: at=info method=GET path=/ host=jonwashere.herokuapp.com
request_id=f6ac74f1-68bf-4cb3-b363-3aa54e5b420f fwd="94.174.204.242" dyno=web.1 connect=2ms service=12ms
status=200 bytes=191
May 13 14:43:09 jonwashere app/web.1: 10.236.149.233 - - [Tue, 13 May 2014 21:43:08 GMT] "GET / HTTP/1.1" 200
13 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_2) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/34.0.1847.131 Safari/537.36"
May 13 14:43:29 jonwashere heroku/router: at=info method=GET path=/favicon.ico host=jonwashere.herokuapp.com
request_id=51f36ddf-9b81-4f54-ae5f-f17573d30e4a fwd="94.174.204.242" dyno=web.1 connect=0ms service=6ms
status=404 bytes=193
```

## Start a console

You can run a command, typically scripts and applications that are part of your app, in a one-off dyno (<https://devcenter.heroku.com/articles/one-off-dynos>) using the `heroku run` command. It can also be used to launch a REPL process attached to your local terminal for experimenting in your app's environment:

```
$ heroku run python manage.py shell
Python 3.7.6 (default, Dec 23 2019, 04:25:22)
[GCC 7.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>>
```

If you receive an error, `Error connecting to process`, then you may need to configure your firewall (<https://devcenter.heroku.com/articles/one-off-dynos#timeout-awaiting-process>).

The Python shell is running in the context of your app and all its dependencies. From here you can import some of your application files. For example, you will be able to run the following:

```
>>> import requests
>>> print(requests.get('http://httpbin.org/status/418').text)

--[ teapot ]==

      .-..-.
     /   _ \
    |  ^__^
    | (oo)\_____)
    | (__|       )\/\
    ||----w |
    ||     ||
    \_____.-.-./
     \___/_/

>>> exit()
```

To get a real feel for how dynos work, you can create another one-off dyno and run the `bash` command, which opens up a shell on that dyno. You can then execute commands there. Each dyno has its own ephemeral filesystem, populated with your app and its dependencies - once the command completes (in this case, `bash`), the dyno is removed.

```
$ heroku run bash
Running `bash` attached to terminal... up, run.3052
~ $ ls
gettingstarted  hello  manage.py  Procfile  README.md  requirements.txt  runtime.txt  staticfiles
~ $ exit
exit
```

Don't forget to type `exit` to exit the shell and terminate the dyno.

## Define config vars

Heroku lets you externalise configuration - storing data such as encryption keys or external resource addresses in config vars (<https://devcenter.heroku.com/articles/config-vars>).

At runtime, config vars are exposed as environment variables to the application.

Edit `hello/views.py`. At the beginning, add a line to import the `os` module:

```
import os
```

Now modify the `index` method so that it repeats an action depending on the value of the `TIMES` environment variable:

```
def index(request):
    times = int(os.environ.get('TIMES',3))
    return HttpResponse('Hello! ' * times)
```

heroku local will automatically set up the environment based on the contents of the .env file in your local directory. In the top-level directory of your project there is already a .env file that has the following contents:

If you run the app with `heroku local`, you'll see two "Hello!"s.

To set the config var on Heroku, execute the following:

```
$ heroku config:set TIMES=2
```

View the config vars that are set using `heroku config :`

```
$ heroku config
== sharp-rain-871 Config Vars
PAPERTRAIL_API_TOKEN: erdKhPeeehIcdfY7ne
TIMES: 2
```

Deploy your changed application to Heroku to see this in action.

## Provision a database

The add-on marketplace (<https://elements.heroku.com/addons/categories/data-stores>) has a large number of data stores, from Redis and MongoDB providers, to Postgres and MySQL. In this step you will learn about the free Heroku Postgres add-on that was automatically provisioned when your app was deployed.

A database is an add-on, and so you can find out a little more about the database provisioned for your app using the `addons` command in the CLI:

```
$ heroku addons
Add-on                               Plan    Price  State
-----
heroku-postgresql (postgresql-horizontal-27446) hobby-dev  free   created
└─ as DATABASE
...
```

Listing the config vars for your app will display the URL that your app is using to connect to the database, `DATABASE_URL` :

```
$ heroku config
=== serene-caverns-82714 Config Vars
DATABASE_URL: postgres://qayojflkqzwdlk:c9b49b89f95625e0c1ed225aed3871a888ab41ca53d6d16078fe5f6416f8a402@ec2-23-23-80-20.compute-1.amazonaws.com:543
```

Heroku also provides a `pg` command that shows a lot more:

```
$ heroku pg
=== DATABASE_URL
Plan:           Hobby-dev
Status:         Available
Connections:    0/20
PG Version:     10.5
Created:        2018-10-12 19:13 UTC
Data Size:      7.6 MB
Tables:         0
Rows:           0/10000 (In compliance) - refreshing
Fork/Follow:    Unsupported
Rollback:       Unsupported
Continuous Protection: Off
Add-on:         postgresql-horizontal-27446
```

This indicates I have a hobby database (free), running Postgres 10.5, with no data.

The example app you deployed already has database functionality, which you should be able to reach by visiting your app's URL and appending `/db` . For example, if your app was deployed to `https://wonderful-app-287.herokuapp.com/` then visit `https://wonderful-app-287.herokuapp.com/db` .

Accessing it will yield an error though, because while the database is configured, the tables have not been created. Run the standard Django `manage.py migrate` to create the tables.

```
$ heroku run python manage.py migrate
Running 'python manage.py migrate' attached to terminal... up, run.1059
Synchronizing apps without migrations:
  Creating tables...
    Creating table hello_greeting
    Running deferred SQL...
  Installing custom SQL...
Running migrations:
  Rendering model states... DONE
  Applying contenttypes.0001_initial... OK
...
```

If you see a message that says, "You just installed Django's auth system, which means you don't have any superusers defined. Would you like to create one now?", type `no` .

Now access the `/db` route again and you'll see a simple page update every time you access it:

```
Page View Report

April 19, 2017, 8:50 a.m.
April 19, 2017, 8:52 a.m.
```

The code to access the database is straightforward, and makes use of a simple Django model called `Greetings` that you can find in `hello/models.py` .

Whenever you visit the `/db` route of your app, the following method in the `hello/views.py` file is invoked which creates a new `Greeting` and then renders all the existing `Greetings`:

```
def db(request):

    greeting = Greeting()
    greeting.save()

    greetings = Greeting.objects.all()

    return render(request, 'db.html', {'greetings': greetings})
```

Assuming that you have Postgres installed locally (<https://devcenter.heroku.com/articles/heroku-postgresql#local-setup>), use the `heroku pg:psql` command to connect to the remote database and see all the rows:

```
$ heroku pg:psql
--> Connecting to postgresql-horizontal-27446
psql (10.5)
SSL connection (protocol: TLSv1.2, cipher: ECDHE-RSA-AES256-GCM-SHA384, bits: 256, compression: off)
Type "help" for help.

serene-caverns-82714::DATABASE=>
```

Read more about Heroku PostgreSQL (<https://devcenter.heroku.com/articles/heroku-postgresql>).



A similar technique can be used to install MongoDB or Redis add-ons (<https://elements.heroku.com/addons/categories/data-stores>).

## Next steps

You now know how to deploy an app, change its configuration, view logs, scale, and attach add-ons.

Here's some recommended reading. The first, an article, will give you a firmer understanding of the basics. The second is a pointer to the main Python category here on Dev Center:

- Read *How Heroku Works* (<https://devcenter.heroku.com/articles/how-heroku-works>) for a technical overview of the concepts you'll encounter while writing, configuring, deploying and running applications.
- Read *Deploying Python and Django Apps on Heroku* (<https://devcenter.heroku.com/articles/deploying-python>) to understand how to take an existing Python or Django app and deploy it to Heroku.
- Visit the Python category (<https://devcenter.heroku.com/categories/python-support>) to learn more about developing and deploying Python applications.