Getting Started on Heroku with Python

English — 日本語に切り替える (/ja/articles/getting-started-with-python)

Last updated April 29, 2022

Table of Contents

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 3.10 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:

macOS

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

Also available via Homebrew:

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

Windows

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)

Ubuntu 16+

Run the following from your terminal:

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

Snap is available on other Linux OS's as well (https://snapcraft.io).

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

On Windows, start the Command Prompt (cmd.exe) or Powershell to access the 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.

Before continuing, make sure Git is installed (see Set up).

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 which Python version will be used, 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.

Before continuing, make sure both Git and the Heroku CLI are installed (see Set up).

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
Enumerating objects: 509, done.
Counting objects: 100% (509/509), done.
Delta compression using up to 12 threads
Compressing objects: 100% (238/238), done.
Writing objects: 100% (509/509), 94.80 KiB | 94.80 MiB/s, done.
Total 509 (delta 234), reused 509 (delta 234), pack-reused 0
remote: Compressing source files... done.
remote: Building source:
remote:
remote: -----> Building on the Heroku-20 stack
remote: -----> Determining which buildpack to use for this app
remote: -----> Python app detected
remote: -----> Using Python version specified in runtime.txt
remote: -----> Installing python-3.10.3
remote: -----> Installing pip 21.3.1, setuptools 57.5.0 and wheel 0.37.0
remote: -----> Installing SQLite3
remote: -----> Installing requirements with pip
remote:        Collecting django
remote:          Downloading Django-4.0.1-py3-none-any.whl (8.0 MB)
remote:        Collecting gunicorn
remote:          Downloading gunicorn-20.1.0-py3-none-any.whl (79 kB)
remote:        Collecting django-heroku
remote:          Downloading django_heroku-0.3.1-py2.py3-none-any.whl (6.2 kB)
remote:        Collecting asgiref<4,>=3.4.1
remote:          Downloading asgiref-3.4.1-py3-none-any.whl (25 kB)
remote:        Collecting sqlparse>=0.2.2
remote:          Downloading sqlparse-0.4.2-py3-none-any.whl (42 kB)
remote:        Collecting psycopg2
remote:          Downloading psycopg2-2.9.3.tar.gz (380 kB)
remote:          Preparing metadata (setup.py): started
remote:          Preparing metadata (setup.py): finished with status 'done'
remote:        Collecting dj-database-url>=0.5.0
remote:          Downloading dj_database_url-0.5.0-py2.py3-none-any.whl (5.5 kB)
remote:        Collecting whitenoise
remote:          Downloading whitenoise-5.3.0-py2.py3-none-any.whl (19 kB)
remote:        Building wheels for collected packages: psycopg2
remote:          Building wheel for psycopg2 (setup.py): started
remote:          Building wheel for psycopg2 (setup.py): finished with status 'done'
remote:          Created wheel for psycopg2: filename=psycopg2-2.9.3-cp310-cp310-linux_x86_64.whl size=586593 sha256=25ea6b40570d00fb7329c9691e557245a105085732389
remote:          Stored in directory: /tmp/pip-ephem-wheel-cache-d5wzowyy/wheels/81/b6/3d/091aad3e8919ea76c84c2674b02ce3ab52de882e091c39249e
remote:        Successfully built psycopg2
remote:        Installing collected packages: sqlparse, asgiref, whitenoise, psycopg2, django, dj-database-url, gunicorn, django-heroku
remote:        Successfully installed asgiref-3.4.1 dj-database-url-0.5.0 django-4.0.1 django-heroku-0.3.1 gunicorn-20.1.0 psycopg2-2.9.3 sqlparse-0.4.2 whitenois
remote: -----> $ python manage.py collectstatic --noinput
remote:        129 static files copied to '/tmp/build_a5ccf75d/staticfiles', 379 post-processed.
remote:
remote: -----> Discovering process types
remote:        Procfile declares types -> web
remote:
remote: -----> Compressing...
remote:        Done: 68.1M
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
```

If when running the `ps:scale` command above you see the error "`Couldn't find that process type (web)`", it means the first deploy of the application is taking a bit longer than normal, and the command should be repeated after waiting a few minutes.

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
2022-01-19T16:49:40.323576+00:00 heroku[web.1]: Starting process with command `gunicorn gettingstarted.wsgi`
2022-01-19T16:49:41.952853+00:00 app[web.1]: [2022-01-19 16:49:41 +0000] [4] [INFO] Starting gunicorn 20.1.0
2022-01-19T16:49:41.953307+00:00 app[web.1]: [2022-01-19 16:49:41 +0000] [4] [INFO] Listening at: http://0.0.0.0:5021 (4)
2022-01-19T16:49:41.954552+00:00 app[web.1]: [2022-01-19 16:49:41 +0000] [4] [INFO] Using worker: sync
2022-01-19T16:49:41.962918+00:00 app[web.1]: [2022-01-19 16:49:41 +0000] [9] [INFO] Booting worker with pid: 9
2022-01-19T16:49:41.992997+00:00 app[web.1]: [2022-01-19 16:49:41 +0000] [10] [INFO] Booting worker with pid: 10
2022-01-19T16:49:42.380291+00:00 heroku[web.1]: State changed from starting to up
2022-01-19T16:49:44.000000+00:00 app[api]: Build succeeded
2022-01-19T16:56:43.488176+00:00 heroku[router]: at=info method=GET path="/" host=serene-caverns-82714.herokuapp.com request_id=df14c065-0d94-45b7-b21e-6e69a0ad93
```

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
```

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 50m (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 2022/01/19 16:49:42 +0000 (~ 3m 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
Scaling dynos... done, now running web at 0:Free
```

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
Scaling dynos... done, now running web at 1:Free
```

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

Install app dependencies locally

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 install the dependencies locally, we first want to create a “Virtual Environment” (also known as a “venv”) into which we can install the packages without affecting your system Python installation. To do this, run the following command:

```
$ python3 -m venv venv
```

Next, the virtual environment needs to be activated.

If you’re on a **Microsoft Windows system**, activate the venv using:

```
.\venv\Scripts\activate
```

Or if you’re on a **macOS/Linux system**, activate the venv using:

```
source venv/bin/activate
```

For help with setting up a virtual environment, see the Python documentation (https://packaging.python.org/en/latest/guides/installing-using-pip-and-virtual-environments/#creating-a-virtual-environment).

Finally, the dependencies can now be installed into the newly created environment:

```
$ 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. 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
--------------- -------
asgiref         3.4.1
dj-database-url 0.5.0
Django          4.0.1
django-heroku   0.3.1
gunicorn        20.1.0
pip             21.3.1
psycopg2        2.9.3
setuptools      59.5.0
sqlparse        0.4.2
wheel           0.37.0
whitenoise      5.3.0
```

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

Run the app locally

Before continuing, make sure the app’s dependencies have been installed 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 a **Microsoft Windows system**, run this:

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

Or if you’re on a **macOS/Linux system**, use the default `Procfile` by running:

```
$ heroku local
```

Your local web server will then start up:

```
[OKAY] Loaded ENV .env File as KEY=VALUE Format
17:39:12 web.1   |  [2022-01-19 17:39:12 +0000] [18471] [INFO] Starting gunicorn 20.1.0
17:39:12 web.1   |  [2022-01-19 17:39:12 +0000] [18471] [INFO] Listening at: http://0.0.0.0:5000 (18471)
17:39:12 web.1   |  [2022-01-19 17:39:12 +0000] [18471] [INFO] Using worker: sync
17:39:12 web.1   |  [2022-01-19 17:39:12 +0000] [18472] [INFO] Booting worker with pid: 18472
```

If you see “Connection in use” errors, check that no other running programs are using port 5000. On macOS Monterey, Airplay Receiver runs on port 5000 and will need to be disabled by going to System Preferences -> Sharing and then unticking “Airplay Receiver”.

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.

If you see “Not Found” errors in your console, check that the `collectstatic` step above was run before starting the web server.

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.

Add the `requests` package to your `requirements.txt` file:

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

And then use pip to install the `requests` package via the updated `requirements.txt` file:

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

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('https://httpbin.org/status/418')
    print(r.text)
    return HttpResponse('<pre>' + r.text + '</pre>')
```

Now test again locally.

If you’re on a **Microsoft Windows system**, run this:

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

Or if you’re on a **macOS/Linux system**, use the default `Procfile` by running:

```
$ 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 ]=-

       _...._
     .'  _ _ `.
    | ."` ^ `". _,
    \_;`"---"`|//
      |       ;/
      \_     _/
        `"""`
```

If instead you see the error `ModuleNotFoundError: No module named 'requests'`, check that the steps to install the `requests` package above were completed successfully.

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.

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), to add a credit-card to your account. This credit card will only be charged if you use paid Heroku features - the default Papertrail plan is free.

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

```
$ heroku addons:create papertrail
Creating papertrail on ⬢ serene-caverns-82714... free
Welcome to Papertrail. Questions and ideas are welcome (technicalsupport@solarwinds.com). Happy logging!
Created papertrail-convex-88929 as PAPERTRAIL_API_TOKEN
Use heroku addons:docs papertrail to view documentation
```

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:

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
Running python manage.py shell on ⬢ serene-caverns-82714... up, run.9594 (Free)
Python 3.10.3 (main, Jan 14 2022, 23:41:42) [GCC 9.3.0] 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 be able to run the following:

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

    -=[ teapot ]=-

       _...._
     .'  _ _ `.
    | ."` ^ `". _,
    \_;`"---"`|//
      |       ;/
      \_     _/
        `"""`
```

To exit the Python shell, run the command `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 filespace, populated with your app and its dependencies - once the command completes (in this case, `bash`), the dyno is removed.

```
$ heroku run bash
Running bash on ⬢ serene-caverns-82714... up, run.3789 (Free)
~ $ ls
app.json  gettingstarted  hello  manage.py  Procfile  Procfile.windows  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:

```
TIMES=2
```

If you run the app with `heroku local` and then visit http://localhost:5000 (http://localhost:5000), 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
=== serene-caverns-82714 Config Vars
DATABASE_URL:         postgres://aadoybxpeabguh:<PASSWORD>@ec2-3-216-113-109.compute-1.amazonaws.com:5432/d8pcm30gp3rago
PAPERTRAIL_API_TOKEN: <SECRET_TOKEN>
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-cubed-97628)  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://aadoybxpeabguh:<PASSWORD>@ec2-3-216-113-109.compute-1.amazonaws.com:5432/d8pcm30gp3rago
...
```

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:            13.5
Created:               2022-01-19 16:49 UTC
Data Size:             7.9 MB/1.00 GB (In compliance)
Tables:                0
Rows:                  0/10000 (In compliance) - refreshing
Fork/Follow:           Unsupported
Rollback:              Unsupported
Continuous Protection: Off
Add-on:                postgresql-cubed-97628
```

This indicates I have a hobby database (free), running Postgres 13.5, 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`.

However, accessing it now will yield a `relation "hello_greeting" does not exist` error, because while the database is configured, the tables have not yet been created.

To create the `hello_greeting` table, run the standard Django `manage.py migrate` command:

```
$ heroku run python manage.py migrate
Running python manage.py migrate on ⬢ serene-caverns-82714... up, run.4231 (Free)
System check identified some issues:

WARNINGS:
hello.Greeting: (models.W042) Auto-created primary key used when not defining a primary key type, by default 'django.db.models.AutoField'.
    HINT: Configure the DEFAULT_AUTO_FIELD setting or the AppConfig.default_auto_field attribute to point to a subclass of AutoField, e.g. 'django.db.models.BigAu
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, hello, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
...
```

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

```
Page View Report

Jan. 19, 2022, 6:12 p.m.
Jan. 19, 2022, 6:17 p.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-cubed-97628
psql (14.1, server 13.5 (Ubuntu 13.5-2.pgdg20.04+1))
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_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.
- Learn more about the Heroku developer experience and CI/CD features in the Heroku Enterprise Developer Learning Journey (https://heroku-journeys.herokuapp.com/).