# Learning Objectives: Preparing for Heroku

**Learners will be able to…**

- **Define platform as a service**
- **Identify the purpose of Gunicorn, the Procfile, and the runtime file**
- **Create a requirements file that can be read by pip**
- **Push a Django project to Heroku**

info

## Make Sure You Know

You are comfortable with basic terminal commands.

## Limitations

Heroku no longer offers free accounts. We do not expect you to purchase an account. This assignment describes the process of using Heroku. You may not be able to follow along for all of it.

# What Is Heroku?

## Heroku

Start by starting the `django` virtual environment.

```
conda activate django
```

warning

### No More Free Accounts

Heroku no longer offers free accounts on their platform. We **do not** expect you to purchase a subscription in order to complete this assignment. What follows is an example of how you would get your project up and running on Heroku.

If we want our Django project running on the internet, we have two choices. One is to run it on our own server. We would need a dedicated machine that runs 24 hours a day. We would be responsible for everything: buying the machine, maintaining the machine, etc. The second option is to use a platform as a service (PaaS) company to host our site for us.

Heroku is a popular PaaS company that eases the burden for deploying websites. This means Heroku builds out and maintains an infrastructure that customers can use to host their own applications. They provide the servers, they provide an operating system, they offer a set of installable software, they provide software updates to ensure a secure system, they generate a URL for our Django app, etc. We are going to discuss how you would prepare a Django project to run on Heroku.

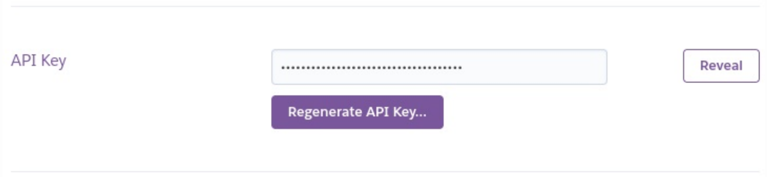
## Creating a Heroku Account

The first thing you would need to do is go to Heroku and create an account. We do not expect you to create a paid account for this activity. This is just a discussion on how to host your Django project on another platform. During the account creation, you may be asked to enable two-factor authentication for your account. So this process could be a bit longer than you would expect.

Once you have our Heroku account, you need to download and install the Heroku command line interface (CLI). Codio runs on the Ubutun operating system, so run the following script in the terminal. Heroku has documentation on how to install the CLI tool on other platforms. You should see a success message after installation.

```
curl https://cli-assets.heroku.com/install-ubuntu.sh | sh
```

You need to login to our Heroku account via the CLI before you can go any further. Heroku will ask for your username and password. However, if your account is set up with multifactor authentication, they will not let you authenticate with this information. Instead, you have to use an API key in place of your password. Go to the tab where you are logged in on Heroku. Click your avatar in the top-right corner. Select account settings. Scroll down until you see the section for API Key. Reveal and copy the API key.



The image depicts the Heroku settings page for your account. Toward the bottom there is a section entitled API Key. The key is represented as a series of dots. To the right of the obscured key is a button to reveal the API key’s contents.

Once you have the key, you can start the login process. In the terminal, enter the following command.

```
heroku login -i
```

You will be prompted for your email and password for Heroku. **Remember**, use the API key in place of your password. The process will look something like the example below.

```
heroku: Enter your login credentials
Email: your-email@provider.com
Password: ************************************
Logged in as your-email@provider.com
```

Next, you would need to create a Heroku application. The name used for the application will also be used in the URL for your Django app. So the application name needs to be unique to you. The command below creates a Heroku application with the specified name. **Important**, replace `<your initials>` with your initials or some other unique identifier.

▼ **Unique names**

> If the name you chose for your app is not unique, you will see a message that this name is already taken. You will have to run `heroku create` again with a different name. You may have to do this a few times until you find a unique name. If you just enter `heroku create`, then Heroku will generate a URL with random words and some numbers.

```
heroku create capital-cities-<your initials>
```

Upon successful application creation, you will see a message like the one below. You get a URL for the Heroku app where you could see your Django app running on the internet. The other URL is for the git repository where our code lives. Take note of the first URL.

```
Creating ⬢ check-birthday... done
https://capital-cities.herokuapp.com/ |
https://git.heroku.com/capital-cities.git
```

We are done working with Heroku for now, so we can logout of our account.

```
heroku logout
```

Deactivate the `django` virtual environment.

```
conda deactivate
```

# Odds and Ends

## Production Web Server

Start the `django` virtual environment.

```
conda activate django
```

When you use the `startproject` for Django, the framework creates most everything we need for our project. In particular, the web server and database Django creates work fine for local development. Unfortunately, both need to be replaced for production. We already talked about using environment variables and Heroku to use PostgreSQL as the production database.

There are a few options out there for the web server. We are going to use Gunicorn as it is fairly simple and straightforward. Start by installing the package from the Conda Forge channel.

```
conda install -c conda-forge gunicorn -y
```

## Procfile

The Procfile is unique to Heroku. It is used to specify any commands you want to run on startup. We need to create this file and, importantly, it must start with a capital `P` and **not** have a file extension. Create the Procfile with the `touch` command.

```
touch Procfile
```

Open the file. We want `gunicorn` to run as our web server. The configurations needed for our server are in the `django_project.wsgi` file, and the `--log-file -` flag means that any logging messages will be visible to us. Add the following command to the Procfile.

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

## Runtime File

Another quirk of Heroku is that you have to specify the version of Python you want to use. This is done through a `runtime.txt` file. Start by opening the terminal and creating this file with the `touch` command.

```
touch runtime.txt
```

Next, we need to determine which version of Python we have been using for development. In the terminal, use the `--version` flag to see which version of Python we have been using.

```
python --version
```

Python will list out the specific version we have been using. **Note**, the version you are using may be different than the one used at the time of writing.

```
Python 3.10.5
```

Open the `runtime.txt` file and enter the version of Python that was listed above. Remember, the version you enter may be different from the one listed here. You **must** use a lowercase `p` and a `-` separates `python` from the version number.

```
python-3.10.5
```

Return to the terminal and deactivate the `django` virtual environment.

```
conda deactivate
```

# Requirements

## Requirements File

Start the `django` virtual environment.

```
conda activate django
```

Even though we have been using Conda for all of our package management, we need to create our requirements file to be compatible with the `pip` package manager, which is the default manager for Python. `pip` is already installed in the `django` virtual environment. However, if we use the typical `freeze` command, we do not get the desired output.

```
pip freeze
```

What we see are all of the package names but there are no version numbers. `pip` cannot use this output to install our dependencies on another system.

```
asgiref @
file:///home/conda/feedstock_root/build_artifacts/asgiref_165275
2042988/work
black @ file:///home/conda/feedstock_root/build_artifacts/black-
recipe_1648499330704/work
click @
file:///home/conda/feedstock_root/build_artifacts/click_16512151
51197/work
dataclasses @
file:///home/conda/feedstock_root/build_artifacts/dataclasses_16
28958434797/work
Django @
file:///home/conda/feedstock_root/build_artifacts/django_1649678
133763/work
environs @
file:///home/conda/feedstock_root/build_artifacts/environs_16417
39190268/work
gunicorn @
file:///home/conda/feedstock_root/build_artifacts/gunicorn_16493
84844586/work
marshmallow @
file:///home/conda/feedstock_root/build_artifacts/marshmallow_16
```

```
61241494193/work
mypy-extensions @
file:///home/conda/feedstock_root/build_artifacts/mypy_extension
s_1649013342054/work
packaging @
file:///home/conda/feedstock_root/build_artifacts/packaging_1637
239678211/work
pathspec @
file:///home/conda/feedstock_root/build_artifacts/pathspec_16266
13672358/work
platformdirs @
file:///home/conda/feedstock_root/build_artifacts/platformdirs_1
645298319244/work
psycopg2 @
file:///home/conda/feedstock_root/build_artifacts/psycopg2-
split_1640944506498/work
pyparsing @
file:///home/conda/feedstock_root/build_artifacts/pyparsing_1652
235407899/work
python-dotenv @
file:///home/conda/feedstock_root/build_artifacts/python-
dotenv_1648162816482/work
sqlparse @
file:///home/conda/feedstock_root/build_artifacts/sqlparse_16313
17292236/work
tomli @
file:///home/conda/feedstock_root/build_artifacts/tomli_16443422
47877/work
typed-ast @
file:///home/conda/feedstock_root/build_artifacts/typed-
ast_1653226030502/work
typing_extensions @
file:///home/conda/feedstock_root/build_artifacts/typing_extensi
ons_1650370875435/work
```

Instead, we are going to list out the dependencies and format them for the typical freeze. Then redirect this output to a requirements file.

```
pip list --format=freeze > requirements.txt
```

Just to be sure, lets look at the contents of the requirement file.

```
cat requirements.txt
```

We should see the name of each dependency followed by a == and the version number.

```
asgiref==3.5.2
black==22.3.0
click==8.1.3
dataclasses==0.8
Django==4.0.4
environs==9.4.0
gunicorn==20.1.0
marshmallow==3.17.1
mypy-extensions==0.4.3
packaging==21.3
pathspec==0.9.0
pip==22.1.2
platformdirs==2.5.1
psycopg2==2.9.3
pyparsing==3.0.9
python-dotenv==0.20.0
setuptools==62.6.0
sqlparse==0.4.2
tomli==2.0.1
typed-ast==1.5.4
typing_extensions==4.2.0
wheel==0.37.1
```

The `dataclasses` package poses a bit of a problem when trying to install on Heroku. We can just remove the dependency. Open the requirements file and remove the line containing `dataclasses`.

Open the terminal and deactivate the `django` virtual environment.

```
conda deactivate
```

# Pushing to Heroku

## Heroku and Git

Start the `django` virtual environment.

```
conda activate django
```

Back when we created a Heroku project with the CLI tool, Heroku created a git repository. We can see this by entering the following command.

```
git remote -v
```

You should see four different lines. Two are for Heroku and two are for GitHub. **Note**, the information you see will be different.

```
heroku  https://git.heroku.com/capital-cities.git (fetch)
heroku  https://git.heroku.com/capital-cities.git (push)
origin  git@github.com:<your github>/capital-cities.git (fetch)
origin  git@github.com:<your github>/capital-cities.git (push)
```

We will use the `git push` command to send our Django project to Heroku. However, we need to be careful about the commands we type. The following command sends our work to our GitHub remote repository:

```
git push origin master
```

While this command sends our work to the Heroku remote repository for hosting:

```
git push heroku master
```

There is a one-word difference separating the commands. Because we are using git, Heroku uses the state of code found in the local repository. Over the last few pages, the state of our code has changed. We have added a `Procfile`, a `runtime.txt`, and a `requirements.txt`. We need to commit these changes to the local repository before pushing to Heroku. Start by adding all of the changed files:

```
git add .
```

Then commit these changes with a message:

```
git commit -m "Add Procfile, runtime.txt, and requirements.txt"
```

Let’s push these changes to the GitHub remote repository so that we have a copy of the latest version of our Django project:

```
git push origin master
```

## Pushing to Heroku

There are a few things we need to before we can push to Heroku. First, login to your Heroku account. Be sure to use the API instead of your password.

```
heroku login -i
```

We need to create a PostgreSQL database on the Heroku server. The `hobby-dev` means this is the lowest tier of product. This is a simple web app, so we do not need anything more robust.

```
heroku addons:create heroku-postgresql:hobby-dev -a capital-
        cities
```

We also need to inform Heroku about the secret key used with our Django project. This information is stored in the `.env` file. Open the file, copy the key, and create the terminal command below. Be sure that you are putting your secret key between the quotes.

```
heroku config:set SECRET_KEY="imDnfLXy-8Y-
        YozfJmP2Rw_81YA_qx1XKl5FeY0mXyY"
```

Our Django project does not have any static files (CSS, JavaScript, images, etc.). We want to disable `collectstatic` on Heroku. The `collectstatic` command collects all of the static files in a Django project and puts them in a single directory. We do not need to do this.

```
heroku config:set DISABLE_COLLECTSTATIC=1
```

We are finally ready to host our project to Heroku. Use the following command to push our code to the Heroku remote repository.

```
git push heroku master
```

However, we are not yet done. The final step is to run the `migrate` command so that our model is stored in the database. Since this action is taking place on a Heroku server, we are going to prepend `heroku run` before our normal Python command.

```
heroku run python manage.py migrate
```

The `open` command will open your browser to the correct Heroku page to see your running project. If Heroku cannot open your browser, it will list the URL that you can copy and paste.

```
heroku open
```

Pushing your project to Heroku does not push your existing SQLite database. The new PostgreSQL database shares the same model, but does not contain any information. You can manually add capital cities to the database using the form.

warning

## Open Website

Hosting this project on another platform means it is open to anyone on the internet. There is no account authorization with this Django project. Anybody could write whatever they want in the form. It is not advisable to leave this site running. The following command will turn off your project on Heroku:

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

▼ **What if I forgot my URL?**

> Go to your Heroku dashboard, which has a list of all your applications. Click on your Flask app. Toward the top-right, click the button that says `Open app`. This will open your Flask app in a new tab.