Django 9: Deployment

IN608: Intermediate Application Development Concepts

Kaiako: Tom Clark & Grayson Orr

Last Session's Content

- Django REST framework
  - Permissions
  - Serialization
  - Viewsets
  - Routers

Today’s Content

- Heroku
  - Heroku CLI
- Postgres on Heroku
- Deploy a Django application to Heroku

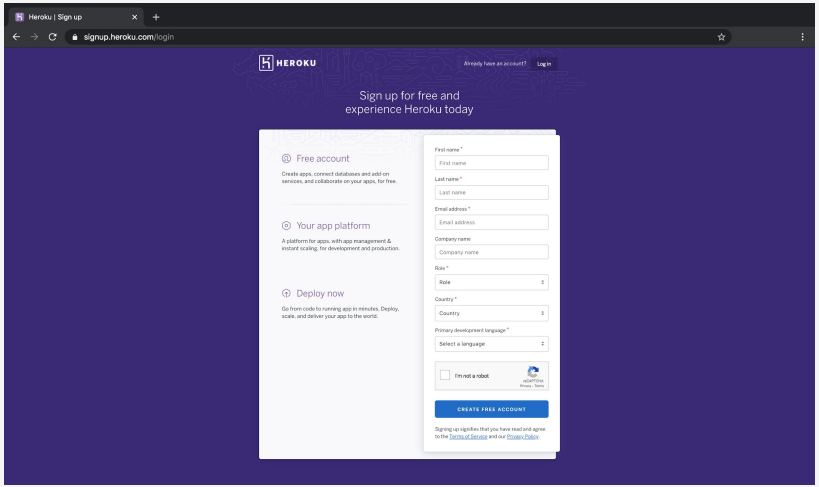
Heroku Deployment

Heroku Deployment

- Heroku lets you build, deploy & manage applications written in a variety of Ruby, Node.js, Java, Python, Clojure, Scala, Go & PHP
- An application's source code & dependency file, i.e., `Pipfile.lock` should provide enough information for the Heroku platform to build your application
- How does Heroku know what command(s) to run?
    - If you are using a framework, i.e., Django or Ruby on Rails, Heroku can figure out what command(s) to run, i.e., `python manage.py runserver` or `rails server`
- You may need to explicitly declare what can be executed. To do this, you need to create a `Procfile`. We will look at this soon

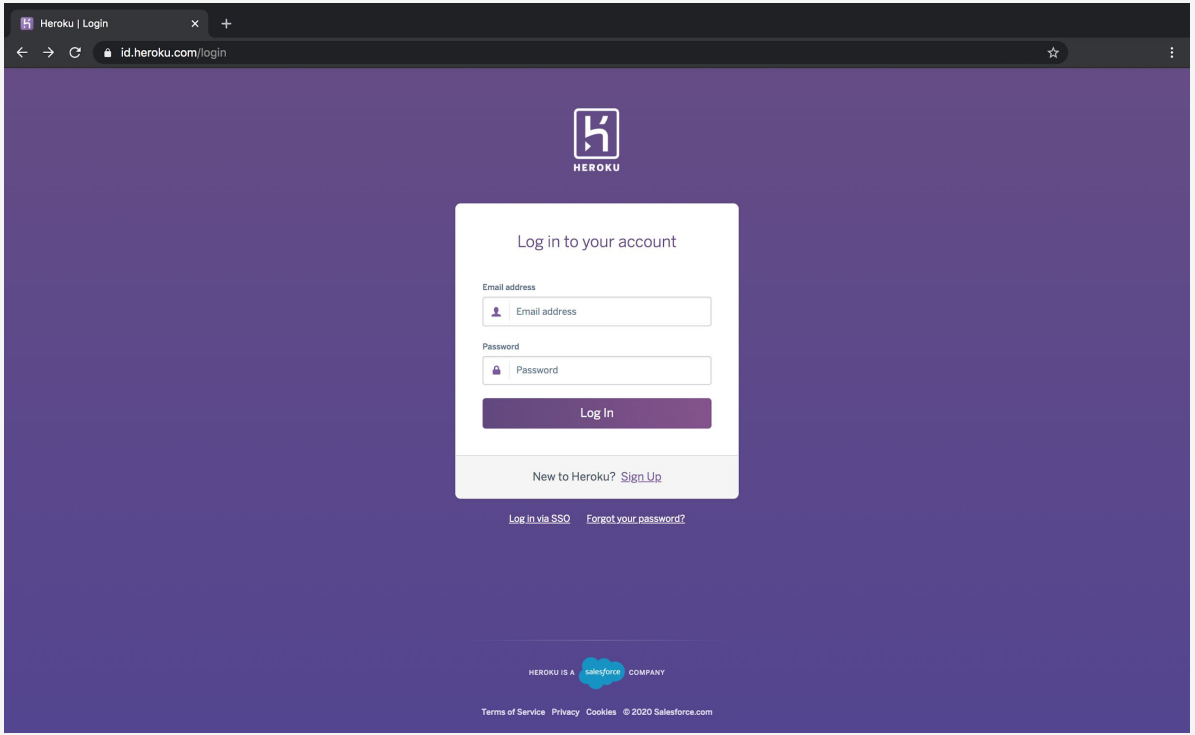
Heroku Signup

- Sign up to Heroku - https://www.heroku.com



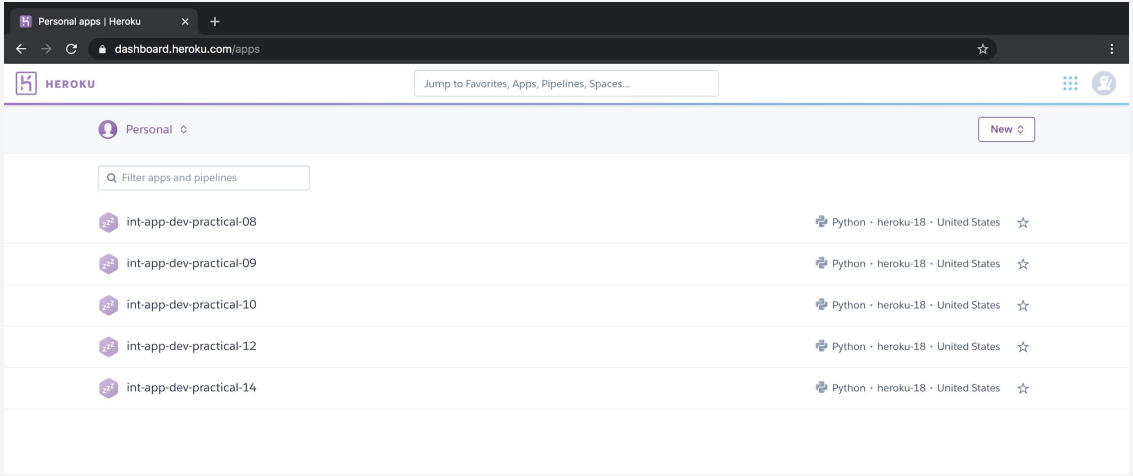
Heroku Signup

- Login to Heroku



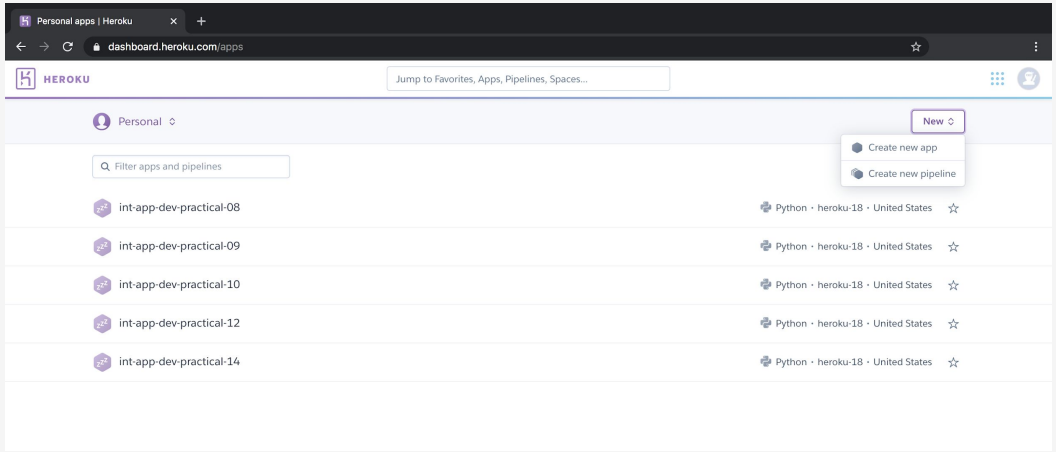
Heroku Dashboard

- Displays all applications



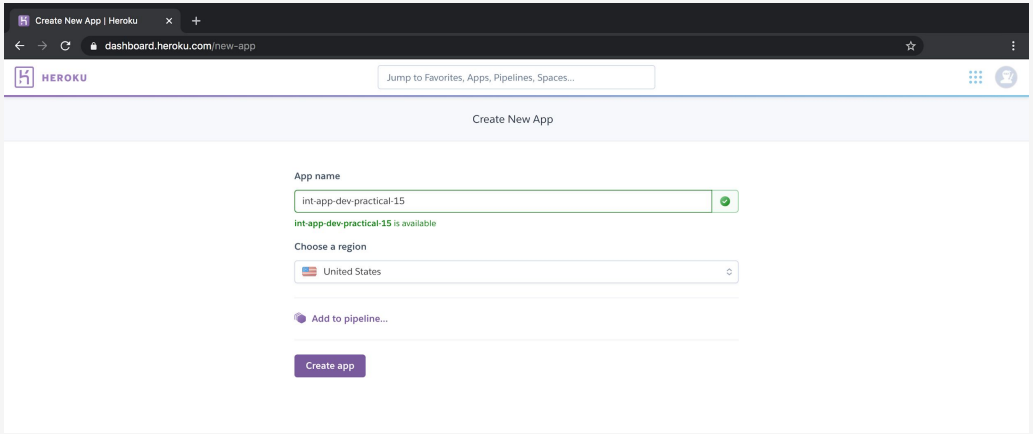
Create an App on Heroku

- New > Create new app
- You can also create a new application using Heroku CLI

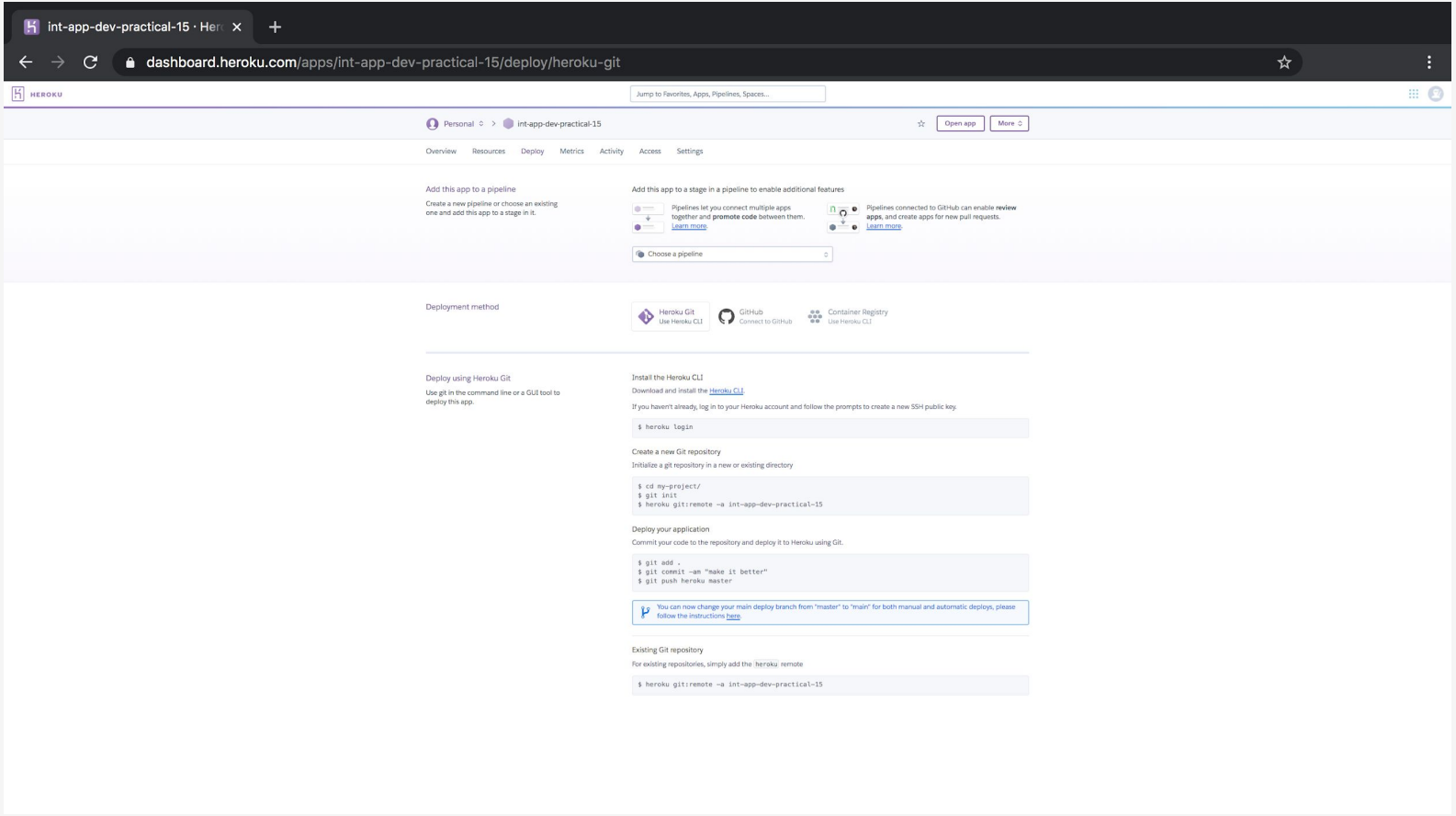


Create an App on Heroku

- Create a new application via GUI



Create an App on Heroku



Heroku CLI

- Heroku CLI (Command Line Interface) makes it easy to build, deploy & manage your Heroku applications from the terminal
- Installation available for macOS, Windows & Linux
  - The Window installer may display a warning - “Windows protected your PC”
  - To run the installation, click “More info”, verify the publisher as “Heroku, Inc.”, then click the “Run anyway” button
- To verify your installation, run the command: `heroku --version`
  - In the output, you should see `heroku/x.x.x`
- After you install Heroku CLI, run the command: `heroku login`
  - You will be prompt to enter any key. This will navigate you to your web browser to complete your login
  - Heroku CLI will then automatically log you in
- Resource: https://devcenter.heroku.com/articles/heroku-cli

Create an App with the Heroku CLI

- Suppose we have been developing a Django app and using Git. From the project's root directory, use the following commands:
- Create an application without a name `heroku create`. A random name will be generated
- Add a remote to your local repository by running the command:
  `heroku git:remote -a <heroku project name>`
- To deploy the application:
  - Add changes in the working directory to the staging area - `git add .`
  - Capture the state of the project at that point of time - `git commit -m "<some message>"`
  - Upload content in the local repository to the remote repository - `git push heroku master`
- To view your application, run the command: `heroku open`

Local Heroku app

- To run our Heroku application locally, run the command: `heroku local:start`
- Navigate http://0.0.0.0:5000/practical15heroku

Heroku Logs

- An application's logs are collected from output streams of all of its running processes, system components & backing services
- Two types of logs:
  - Runtime logs, i.e., application, system, API & add-on logs
  - Build logs - separate from runtime logs while building & deploying your application
- There are various ways to view your logs:
  - Retrieves 100 recent log entries - `heroku logs`
  - `heroku logs --num 200`
  - Displays recent logs & leaves the session open - `heroku logs --tail`
- Resource: https://devcenter.heroku.com/articles/logging

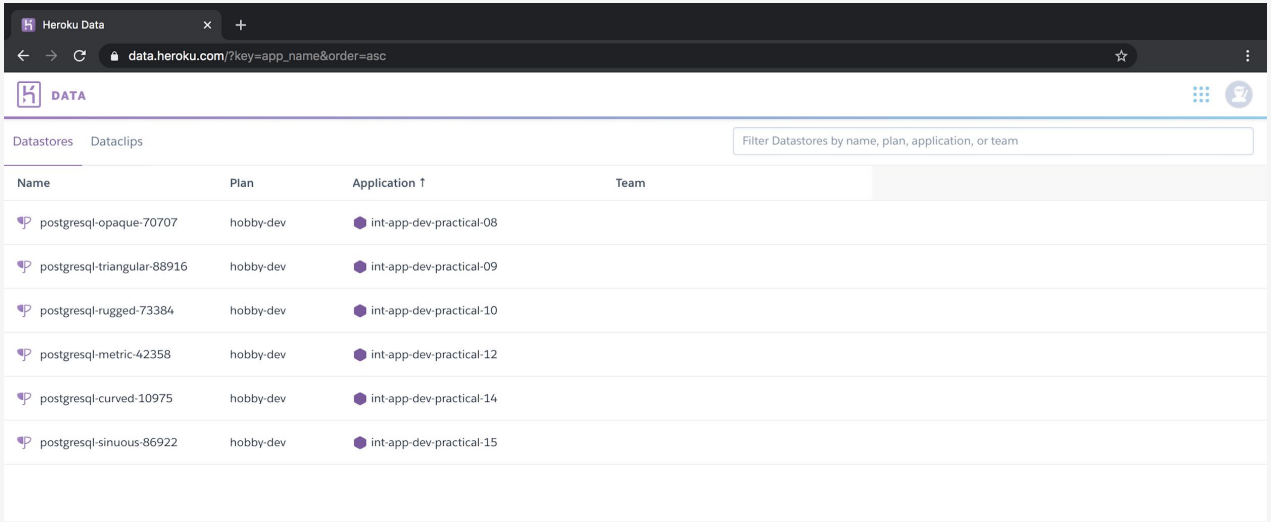
PostgreSQL on Heroku

Postgres on Heroku

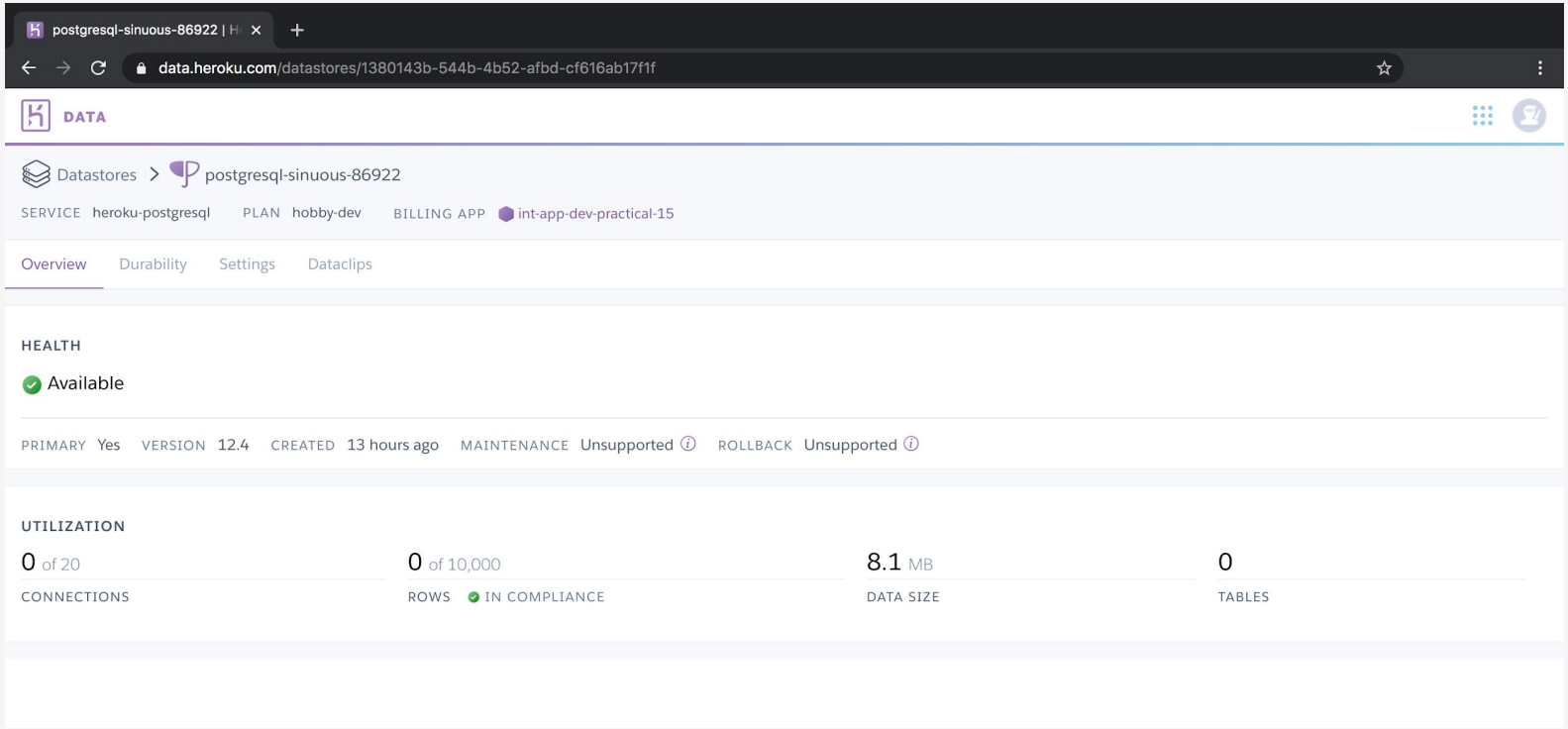
- Heroku Postgres is a managed SQL database service provided directly by Heroku
- Accessible from any language with a PostgreSQL driver
- To check if Heroku Postgres has been provided, run the command: `heroku addons`
- If `heroku-postgresql` is not in the application's list of add-ons, run the command: `heroku addons:create heroku-postgresql:hobby-dev`
- We will need to apply migrations - `heroku run python manage.py migrate`
- Resource: https://devcenter.heroku.com/articles/heroku-postgresql

Postgres on Heroku

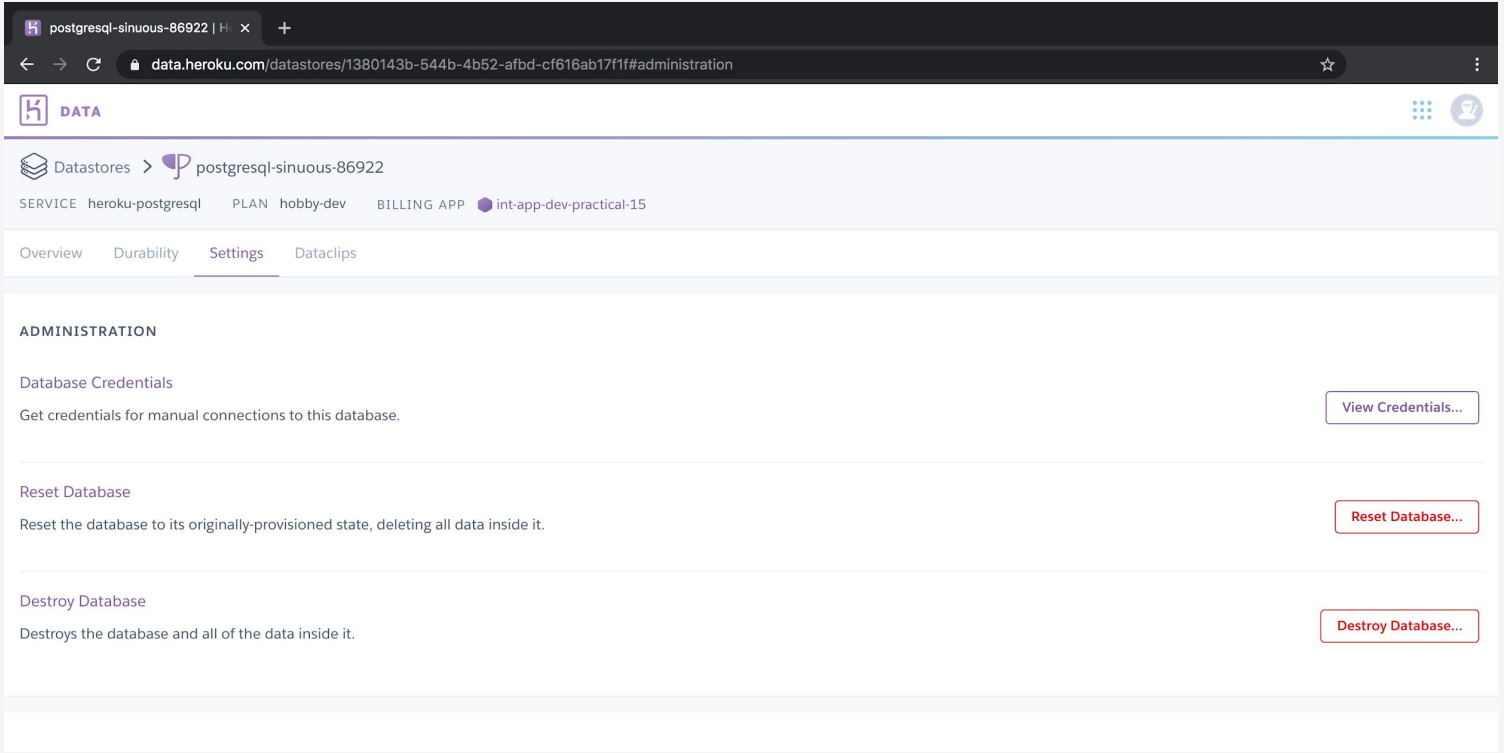
- List of datastores



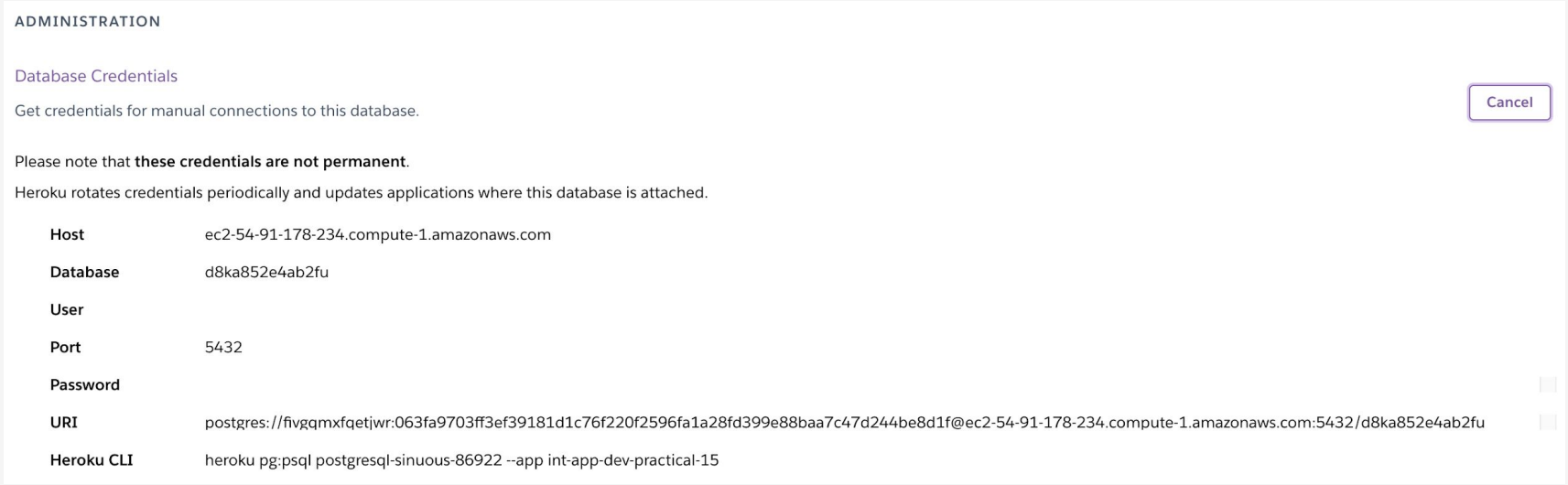
Postgres on Heroku



Postgres on Heroku



Postgres on Heroku



Deploy a Django Application

Deploy a Django Application

We need to make some adjustments to our Django app before it can be run on Heroku

- Create a `.env` file for local development
- Create a config var on Heroku
- Configure database in `settings.py`
- Declare the `STATIC_ROOT` configuration in `settings.py`
- Add `WhiteNoiseMiddleware` to the `MIDDLEWARE` configuration
- Create a `Procfile`

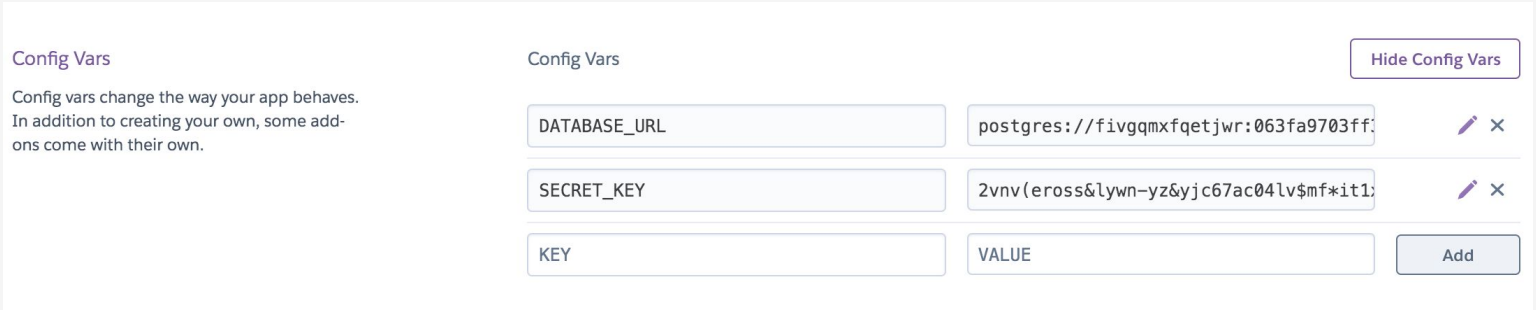
Deploy a Django Application - Settings

- Some settings will differ between development and production versions of our app
- Environment-specific configurations should be stored in environment variables & not in the application's source code, i.e., `SECRET_KEY` in `settings.py`
- In the root directory of your Django project, create a file called `.env`
  - Used for local development
  - Do not store in version control
- Set `SECRET_KEY` in `.env`, i.e., `SECRET_KEY=<some value>`
- Install the dotenv Python module by running the command `pipenv install dotenv`
- To use environment variables locally:
  - In settings.py, declare the following:

```
from dotenv import load_dotenv
load_dotenv()
SECRET_KEY = os.environ.get('SECRET_KEY')
```

Deploy a Django Application - Settings

- Navigate to you Heroku app's settings page
- Heroku sets environments variables using config vars
- The `DATABASE_URL` config variable is automatically created when Heroku Postgres is added
- Set `SECRET_KEY` in config vars
- Resource: https://devcenter.heroku.com/articles/config-vars



Deploy a Django Application - DB Config

How do we connect to Postgres?

- Install the dj-database-url and psycopg2 Python modules
- In settings.py, declare the following:

```
from dj_database_url import config
db_from_env = config(conn_max_age=600)
DATABASES['default'].update(db_from_env) # Put this under the DATABASES config
```

- What is happening?
  - Converts the `DATABASE_URL` config var from Heroku into a Python dictionary
  - The dictionary is injected into the `DATABASES` configuration in `settings.py`
  - We do not have to explicitly set up Heroku Postgres in `settings.py`
- Don't forget to run your migrations: `heroku run python manage.py migrate`

Deploy Django Application - Static Assets

- In Django, static assets can be difficult to configure & debug
- Django does not automatically create `STATIC_ROOT` - the directory in which `collectstatic` uses
- You will need to create this directory so it will be available when `collectstatic` is run
- **Note:** Git does not support empty directories - you will need to create at least one file
- In `settings.py`, declare the following:

```
STATIC_ROOT = os.path.join(BASE_DIR, 'staticfiles')
```

Deploy Django Application - Static Assets

- Django does not support serving static files in production
- `White Noise` is a Python module designed with the purpose of serving static files in production
- To install `White Noise`, run the command: `pipenv install whitenoise`
- Add `whitenoise.middleware.WhiteNoiseMiddleware` to the `MIDDLEWARE` configuration

Deploy a Django Application

- Heroku applications require a `Procfile`
- Explicitly declares an application's process types & entry points
- In the root directory of your Django project, create a file called `Procfile`
- In `Procfile`, declare the following: `web: gunicorn <Django project name>.wsgi`
- This `Procfile` requires `Gunicorn` - recommended production web server for Django applications
- To install `Gunicorn`, run the command: `pipenv install gunicorn`

Programming Activity

- Checkout to master - `git checkout master`
- Create a new branch called 15-practical - `git checkout -b 15-practical`
- Open the file `15-practical.pdf` and work on the tasks described