Web Application Deployment: Render

Robert M. Dondero, Ph.D
Princeton University

Objectives

- The lecture will cover:
  - How to deploy a database and web app to Render
  - Using PostgreSQL
  - Database connection pooling

Agenda

- **Render**
- Creating the DB (demo)
- Using the DB (demo)
- Using the DB: Python
- Defining the app
- Deploying the app (demo)
- Using the app (demo)

Render

Anurag Goel
Founder and CEO

Render

- Render
  - Can create **app** or **DB** first
- Heroku
  - Must create **app** first
  - Then create **DB** specifically for the app

Agenda

- Render
- **Creating the DB (demo)**
- Using the DB (demo)
- Using the DB: Python
- Defining the app
- Deploying the app (demo)
- Using the app (demo)

Creating the DB

- Create a Render account
  - Browse to https://render.com
    - Click *Get Started for Free*
    - Follow the instructions
      - Need not provide a credit card number

Creating the DB

- Log into Render
  - Browse to https://dashboard.render.com/
    - For *Email address* enter `youremailaddress`
    - For *Password* enter `yourpassword`

Creating the DB

- From your Render dashboard
  - Click + *New*
  - Select *Postgres*

Creating the DB

- In the resulting page
  - For *Name* enter `pennydb`
  - For *Region* choose `Ohio (US East)`
  - For *Instance Type* choose `Free`
  - Click *Create Database*
  - Wait (~1 minute) for the database to be created

Creating the DB

- In the resulting page (cont.)
  - Note the *Internal Database URL*
    - **Save it someplace**
    - Let's call it ***YourRenderInternalDbUrl***
  - Note the *External Database URL*
    - **Save it someplace**
    - Let's call it ***YourRenderExternalDbUrl***

Agenda

- Render
- Creating the DB (demo)
- **Using the DB (demo)**
- Using the DB: Python
- Defining the app
- Deploying the app (demo)
- Using the app (demo)

Using the DB

- From a command-line
  - Install `psql`
    - Without installing PostgreSQL server
    - See https://www.risingwave.dev/docs/current/install-psql-without-postgresql/

Using the DB

- From a command-line (cont.)
  - Install `psql` (Mac)
    - First install homebrew; then:

```
$ brew update
$ brew install libpq
$ brew link --force libpq
$
```

Using the DB

- From a command-line (cont.)
  - Install `psql` (MS Windows)
  - Download the installer from https://www.postgresql.org/download/windows/
  - Run the installer
    - Select *Command Line Tools* and uncheck other options during installation

Using the DB

- From a command-line (cont.)
  - Install `psql` (Linux)

```
$ sudo apt update
$ sudo apt install postgresql-client
$
```

Using the DB

Some `psql` statements

sqlite3 Statement	psql Statement
`.help`	`\h`
`.quit`	`\q`
`.tables`	`\d`
`.schema` *table*	`\d` *table*
`.read` *file*	`\i` *file*

Using the DB

```
$ cat penny.sql
DROP TABLE IF EXISTS books;

CREATE TABLE books (isbn TEXT PRIMARY KEY, author TEXT, title TEXT);

INSERT INTO books (isbn, author, title)
   VALUES ('123', 'Kernighan','The Practice of Programming');
INSERT INTO books (isbn, author, title)
   VALUES ('234', 'Kernighan','The C Programming Language');
INSERT INTO books (isbn, author, title)
   VALUES ('345', 'Sedgewick','Algorithms in C');
$
```

Using the DB

```
$ psql YourRenderExternalDbUrl
yourdbname=> \i penny.sql
DROP TABLE
CREATE TABLE
INSERT 0 1
INSERT 0 1
INSERT 0 1
yourdbname=> SELECT * FROM books;
 isbn |   author    |          title
------+-------------+--------------------------
 123  | Kernighan | The Practice of Programming
 234  | Kernighan | The C Programming Language
 345  | Sedgewick | Algorithms in C
(3 rows)

yourdbname=> \q
$
```

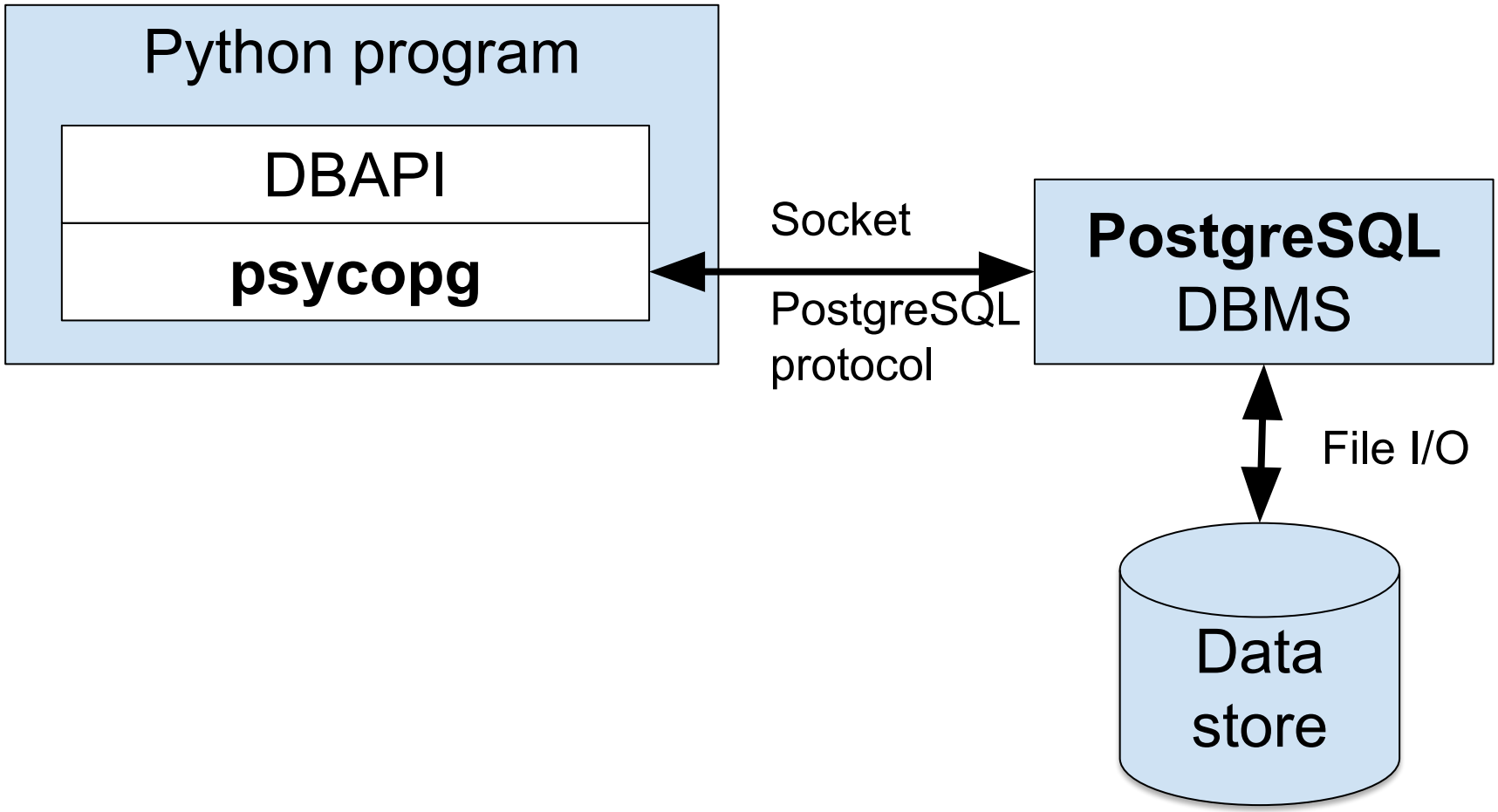
Using the DB

- Graphical PostgreSQL clients:
  - pgAdmin4
  - DBeaver
  - Postico (Mac)
  - …
- List of graphical clients: https://wiki.postgresql.org/wiki/PostgreSQL_Clients

Agenda

- Render
- Creating the DB (demo)
- Using the DB (demo)
- **Using the DB: Python**
- Defining the app
- Deploying the app (demo)
- Using the app (demo)

Using the DB: Python



Using the DB: Python

- Installing the `psycopg` driver:

```
$ activate333
$ python -m pip install psycopg
$
```

Using the DB: Python

- See **pennyrender/databasesqlite.py**
  - Baseline version
  - Uses SQLite

```
$ python databasesqlite.py
123
Kernighan
The Practice of Programming

234
Kernighan
The C Programming Language

$
```

Using the DB: Python

- See **pennyrender/databasepostgres1.py**
  - Uses PostgreSQL
  - Uses an environment variable

```
$ export DATABASE_URL=YourRenderExternalDbUrl
$ python databasepostgres.py
123
Kernighan
The Practice of Programming

234
Kernighan
The C Programming Language

$
```

Using the DB: Python

- **Problem**:
  - Creating PostgreSQL DB connections is *very* slow
- **Attempted solution:**
  - One global DB connection

Using the DB: Python

- See **pennyrender/databasepostgres2bad.py**

```
$ export DATABASE_URL=YourRenderExternalDbUrl
$ python databasepostgres2bad.py
123
Kernighan
The Practice of Programming

234
Kernighan
The C Programming Language

$
```

Using the DB: Python

- **Problem**:
  - Race condition!!!
- **Solution:**
  - ***DB connection pooling***
    - Maintain a “pool” of open DB connections that threads can use

Using the DB: Python

- See **pennyrender/databasepostgres3.py**

```
$ export DATABASE_URL=YourRenderExternalDbUrl
$ python databasepostgres3.py
123
Kernighan
The Practice of Programming

234
Kernighan
The C Programming Language

$
```

Using the DB: Python

- **Alternative solution**:
  - ***SQLAlchemy***

Using the DB: Python

- ***SQLAlchemy***
  - An ***Object Relational Mapper (ORM)*** for Python
    - Maps each DB table to a Python class
    - Maps each DB row to a Python object
  - Works for many popular relational DBMSs
    - SQLite, PostgreSQL, MySQL and MariaDB, Oracle, Microsoft SQL Server
  - See optional lecture for more thorough description

Using the DB: Python

- Installing SQLAlchemy

```
$ activate333
$ python -m pip install SQLAlchemy
$
```

Using the DB: Python

- See **pennyrender/database.py**

```
$ export DATABASE_URL=YourRenderExternalDbUrl
$ python database.py
123
Kernighan
The Practice of Programming

234
Kernighan
The C Programming Language

$
```

Using the DB: Python

- See **pennyrender/database.py** (cont.)
  - Bonus:

```
$ export DATABASE_URL="sqlite:///penny.sqlite"
$ python database.py
123
Kernighan
The Practice of Programming

234
Kernighan
The C Programming Language

$
```

Using the DB: Python

- SQLAlchemy in COS 333 projects?
  - Pros:
    - Hides SQL
    - Portable across many relational DBMSs
    - Automatically uses prepared statements
    - Automatically provides DB connection pooling
    - Allows use of ***Alembic*** (database migration)
    - Harder to make a mistake

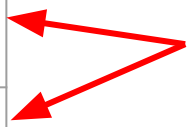
Using the DB: Python

- SQLAlchemy in COS 333 projects?
  - Cons:
    - Hides SQL!!!
    - Must learn SQLAlchemy syntax
    - More complex than SQL
    - Less important than SQL
      - Python only
      - Cannot be used via CLIs (`sqlite3`, `psql`)
    - Poor documentation

Using the DB: Python

For PostgreSQL DB on local computer:

DB SW	Optimization Technique	Time * (sec)	Thread safe?
psycopg	(None)	236.9	Yes
psycopg	One global connection	3.2	No
psycopg	DB connection pooling	6.7	Yes
SQLAlchemy	DB connection pooling	12.2	Yes



Use one of these in your project

* Time to call `get_books()` 10000 times

Aside: PgBouncer

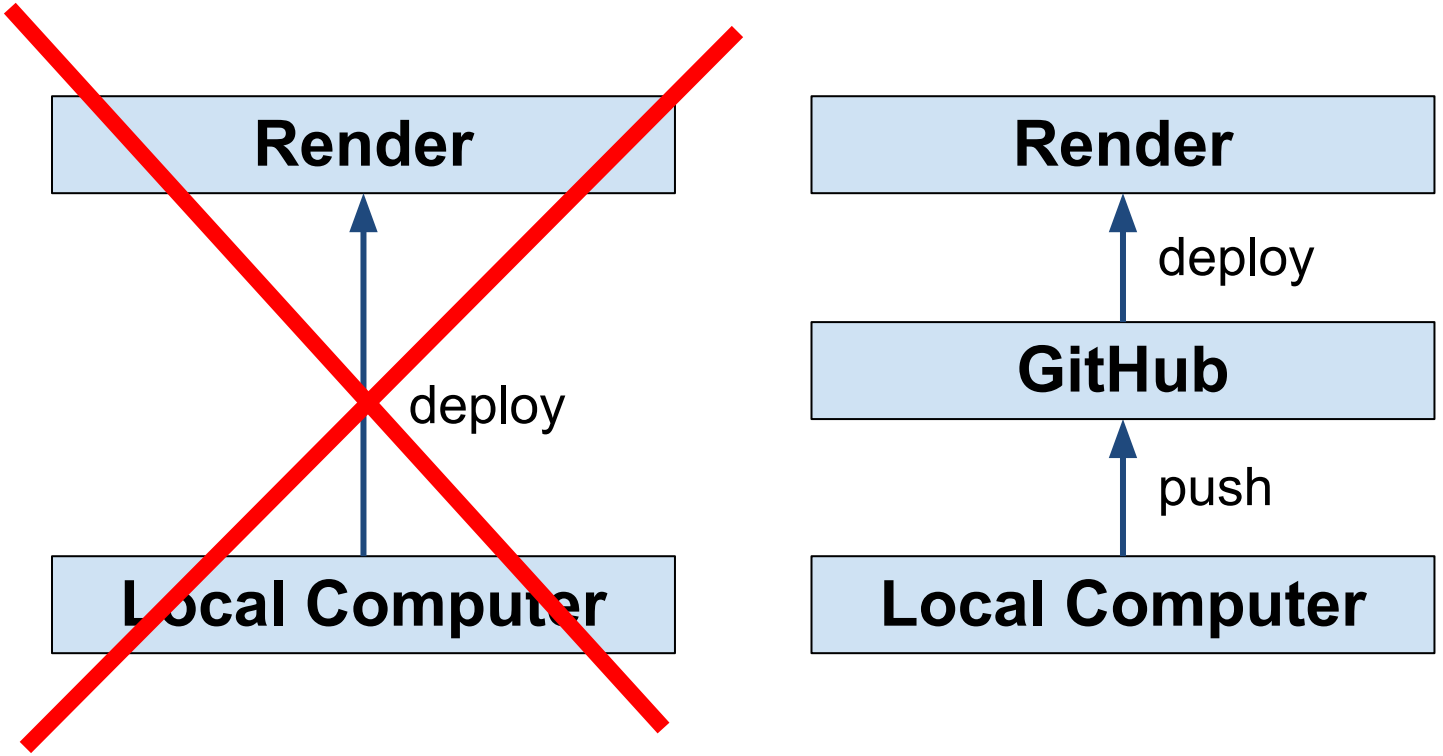
- PostgreSQL connection pool shared by **threads:**
  - Use `psycopg_pool.ConnectionPool`
  - Use SQLAlchemy
- PostgreSQL connection pool shared by **processes**
  - Use ***PgBouncer***
    - Render DB: optional; difficult
    - Neon DB: optional; easy

Agenda

- Render
- Creating the DB (demo)
- Using the DB (demo)
- Using the DB: Python
- **Defining the app**
- Deploying the app (demo)
- Using the app (demo)

Defining the App

- Deployment



Defining the App

- You must change names as appropriate…

Defining the App

- After performing setup steps in *Git and GitHub Primer* document...

- Create a GitHub repo
  - Browse to https://github.com
    - Click the *New* button
      - For *Repository name* enter `pennyrender`
      - Click the *Private* radio button
      - Check the *Add a README file* check button
      - Click the *Create Repository* button

Defining the App

- Clone the GitHub repo to your local computer

```
$ git clone https://github.com/rdondero/pennyrender.git
```

  - Creates `pennyrender` dir on local computer

Defining the App

- Create these files in pennyrender dir:
  - From the PennyFlaskJinja app
    - runserver.py (optionally)
    - penny.sql
    - penny.sqlite (optionally)
    - header.html, footer.html, index.html, searchform.html, searchresults.html
    - penny.py
  - From this lecture
    - database.py

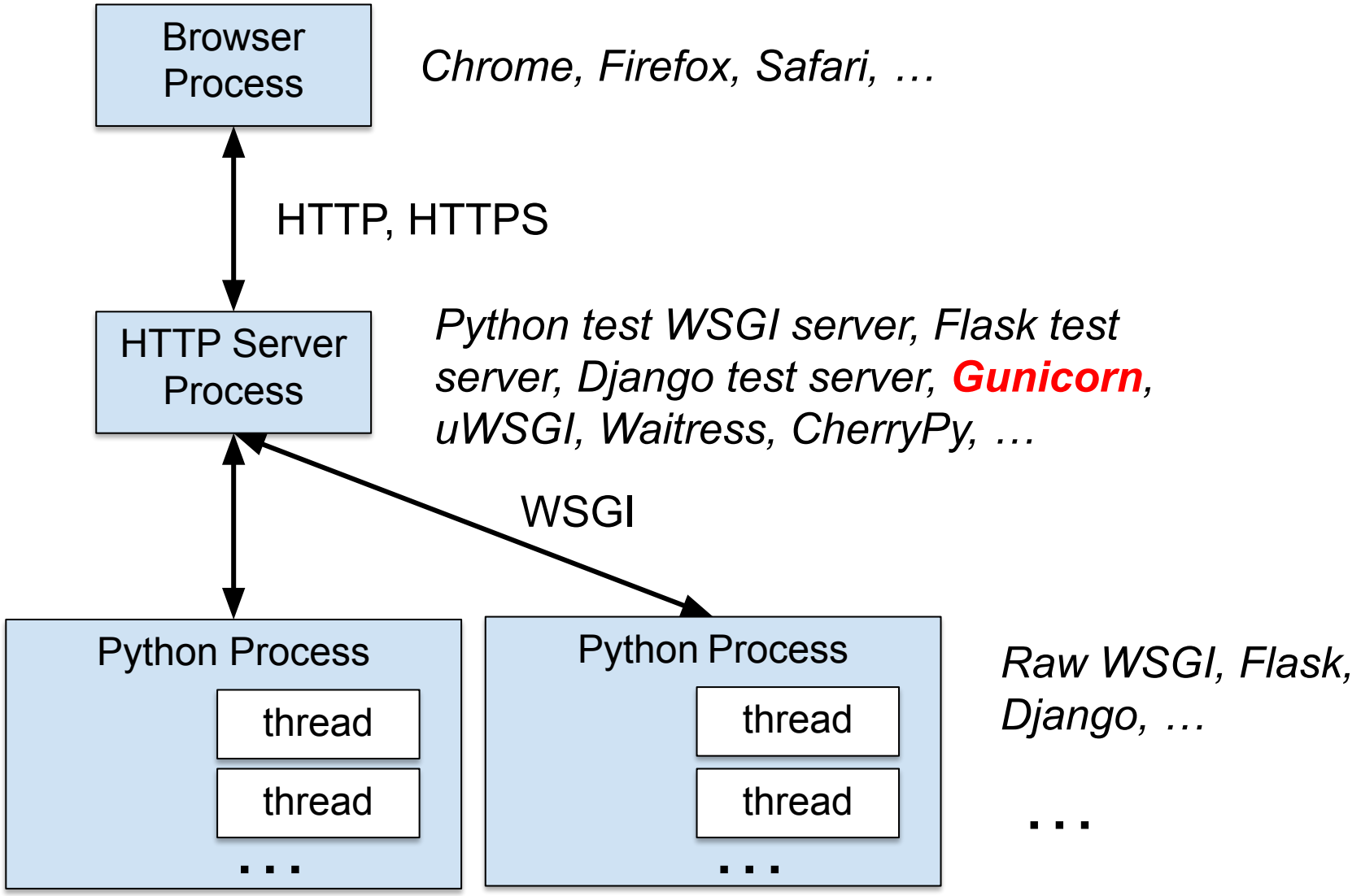
Defining the App

- Also create in pennyrender dir:
  - **requirements.txt**

```
Flask
psycopg
psycopg-pool
SQLAlchemy
python-dotenv
gunicorn
```

    - Tells Render what additional modules the app uses
    - Create manually, or issue command:
      `python -m pip freeze > requirements.txt`

Aside: Gunicorn



Defining the App

- Stage the app to the local git repo, commit the app to the local git repo, and push it to the GitHub repo

```
$ cd pennyrender
$ git add .
$ git commit -m "initial load"
$ git push origin main
```

Agenda

- Render
- Creating the DB (demo)
- Using the DB (demo)
- Using the DB: Python
- Defining the app
- **Deploying the app (demo)**
- Using the app (demo)

Deploying the App

- “Install Render” on GitHub
  - Inform GitHub that Render is permitted to access the repo
  - Browse to https://github.com/apps/render/installations/new
    - Click *rdondero*

Deploying the App

- “Install Render” on GitHub (cont.)
  - In the resulting Render page
    - Click *Only select repositories*
    - Click *Select repositories*
    - Choose *rdondero/pennyrender*
    - Click *Save*

Deploying the App

- (If necessary) Browse to Render dashboard
  - Browse to https://dashboard.render.com
    - Click *Dashboard*

Deploying the App

- Create a new Render app
  - In the *Dashboard* page
    - Click + *New*
    - Select *Web Service*
    - Select *Git Provider*
    - Select rdondero/pennyrender

Deploying the App

- Create a new app (cont.)
  - In the resulting page
    - For *Name* enter **`pennyrender`**
    - For *Region* choose **`Ohio (US East)`**
    - For *Start Command* enter `gunicorn` **`penny:`**`app`
    - For *Instance Type* choose `Free`
    - In *Environment Variables*
      - For *NAME_OF_VARIABLE* enter `DATABASE_URL`
      - For *value* enter *`YourRenderInternalDbUrl`*
    - In *Advanced*
      - Set *Auto-Deploy* to *`Off`*
    - Click *Deploy Web Service*

Deploying the App

- Create a new app (cont.)
  - The app deployment begins
    - Wait up to 5 minutes for deployment to finish

Deploying the App

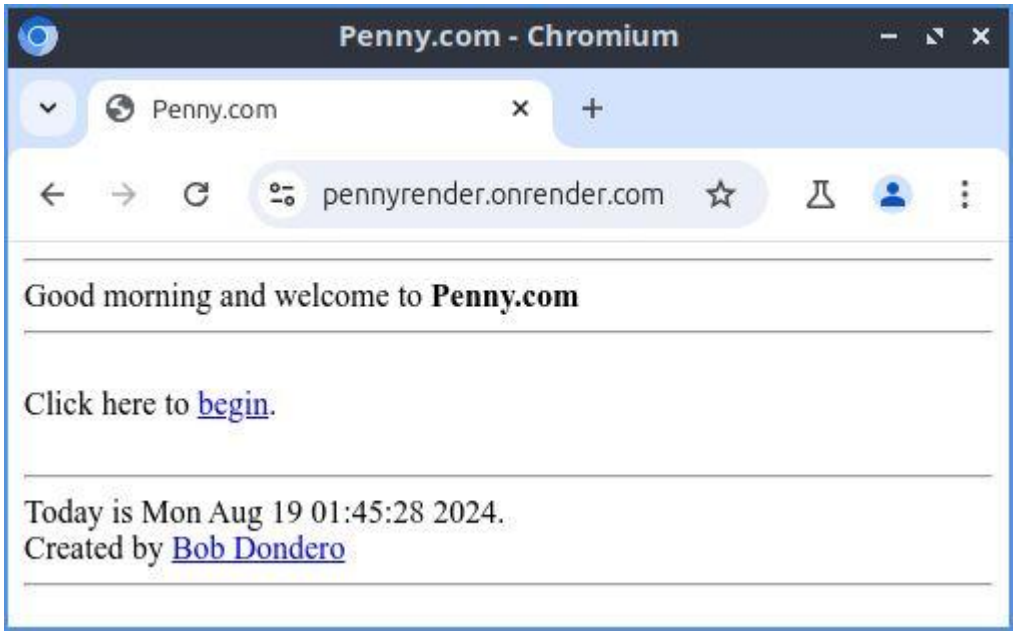
- Thereafter:
  - Push new code from your local git repo to the pennyrender GitHub repo
  - In Render, click *Manual Deploy*
  - Repeat as necessary

Agenda

- Render
- Creating the DB (demo)
- Using the DB (demo)
- Using the DB: Python
- Defining the app
- Deploying the app (demo)
- **Using the app (demo)**

Using the App

- Browse to the app!
  - https://pennyrender.onrender.com



Lecture Summary

- In this lecture we covered:
  - How to deploy a database and web app to Render
  - Using PostgreSQL
  - Database connection pooling
- See also:
  - **Optional lecture**: SQLAlchemy
  - **Optional lecture**: PostgreSQL
  - **Optional lecture:** Web Application Deployment: Heroku
  - **Appendix 1**: PostgreSQL Backup and Restore

Appendix 1:
PostgreSQL Backup and Restore

PostgreSQL Backup and Restore

- **Step 1**
  - (If necessary) install `pg_dump`
    - It probably was installed when you installed `psql`

PostgreSQL Backup and Restore

- **Step 2**
  - Export the Render DB to a text file

```
$ pg_dump YourRenderExternalDbUrl > penny.postgresql
$
```

PostgreSQL Backup and Restore

- **Step 3**
  - Delete the old Render DB
    - Perform through the Render website
- **Step 4**
  - Create a new Render DB
    - Use the instructions provided previously in this lecture

PostgreSQL Backup and Restore

- **Step 5**
  - Import from the text file to the new Render DB

```
$ psql YourNewRenderExternalDbUrl
==> \i penny.postgresql
==> \q
$
```

PostgreSQL Backup and Restore

- **Step 6**
  - Change the value of the `DATABASE_URL` env var of your deployed Render app to *YourNewRenderExternalDbUrl*
- **Step 7**
  - Re-deploy your app to Render