```
from pet_pals.app import db

# db.drop_all()
db.create_all()
```

- In the `pet_pals` directory, there is `app.py`. From it, import `db`, which is a running instance of SQLAlchemy.

```
from flask_sqlalchemy import SQLAlchemy
app.config['SQLALCHEMY_DATABASE_URI'] = os.environ.get('DATABASE_URL', '') or "sqlite:///"
db = SQLAlchemy(app)
```

  - `pet_pals.app` here, then, means `app.py` in `pet_pals`.

- Next, open `models.py`

```
from .app import db

class Pet(db.Model):
    __tablename__ = 'pets'

    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(64))
    lat = db.Column(db.Float)
    lon = db.Column(db.Float)

    def __repr__(self):
        return '<Pet %r>' % (self.name)
```

  - The class model of the SQL table is defined here.
  - Here, too, `db` is imported from `app.py`.
  - Unlike in previous examples, the model for the database has been separated out. This concept is called the separation of concerns: https://stackoverflow.com/questions/98734/what-is-separation-of-concerns.
  - Separation of concerns leads to better organized code and easier troubleshooting.

- In `app.py`, explain that `DATABASE_URL` would be replaced by the connection string to the cloud database during deployment on Heroku, for example.

```
from flask_sqlalchemy import SQLAlchemy
app.config['SQLALCHEMY_DATABASE_URI'] = os.environ.get('DATABASE_URL', '') or "sqlite:///pets_db.sqlite"
db = SQLAlchemy(app)
```

14. Everyone Do: Heroku Deployment (0:20)

- In this activity, we will deploy the Pet Pals application to Heroku. This step consists of 3 main parts:

  i. Prepare the application with additional configuration files (`Procfile` and `requirements.txt`)
  ii. Create the Heroku application
  iii. Prepare the Heroku database

Part 1: Configuration Files

- If you haven't already, send the code from the previous activity to the class.

- Start by creating a new conda environment just for this app. All of our project dependencies will be installed in this environment. Note: This should only contain python 3.6 and not anaconda.

```
conda create -n pet_pals_env python=3.6
```

- Make sure to activate this new environment before proceeding.

```
source activate pet_pals_env
```

- Next, we install `gunicorn` with `pip install gunicorn`. Explain that gunicorn is a high performance web server that can run their Flask app in a production environment.
- Because this app will use Postgres, we also install `psycopg2` with `pip install psycopg2`.
- Make sure to install any other dependencies that are required by the application. This may be `Pandas`, `flask-sqlalchemy`, or any other Python package that is required to run the app. **Test the app locally to make sure that it works!**

```
pip install gunicorn
pip install psycopg2
pip install flask
pip install flask-sqlalchemy
pip install pandas
```

- Test the app by first initializing the database:

```
python initdb.py
```

- Run the app using the following:

```
FLASK_APP=pet_pals/app.py flask run
```

- Now that all of the the project dependencies are installed, we need to generate the `requirements.txt` file. This file is a list of the Python packages required to run the app, we run `pip freeze > requirements.txt`. Heroku will use this file to install all of the app's dependencies.
- The final configuration file that we need is `Procfile`. This file is used by Heroku to run the app.

```
touch Procfile
```

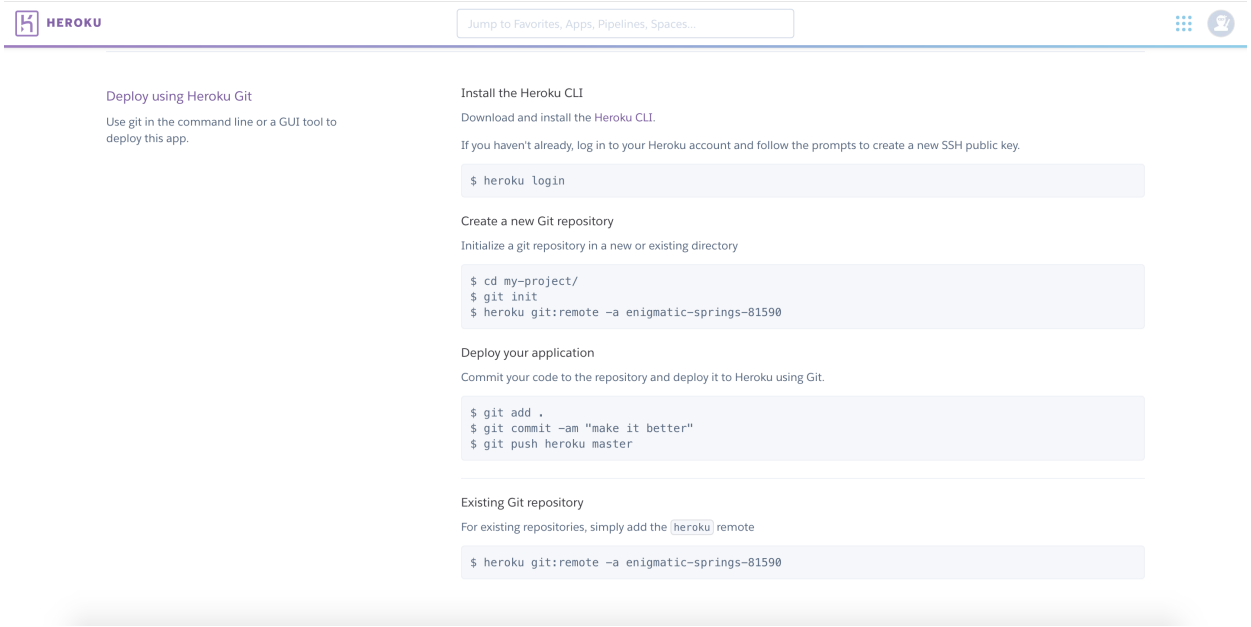
- Open Procfile in vscode and add the following line:

```
web: gunicorn pet_pals.app:app
```

- Explain that `pet_pals` is the name of the folder that contains your app as a python package (i.e. the name of the folder with the `__init__.py` file in it).

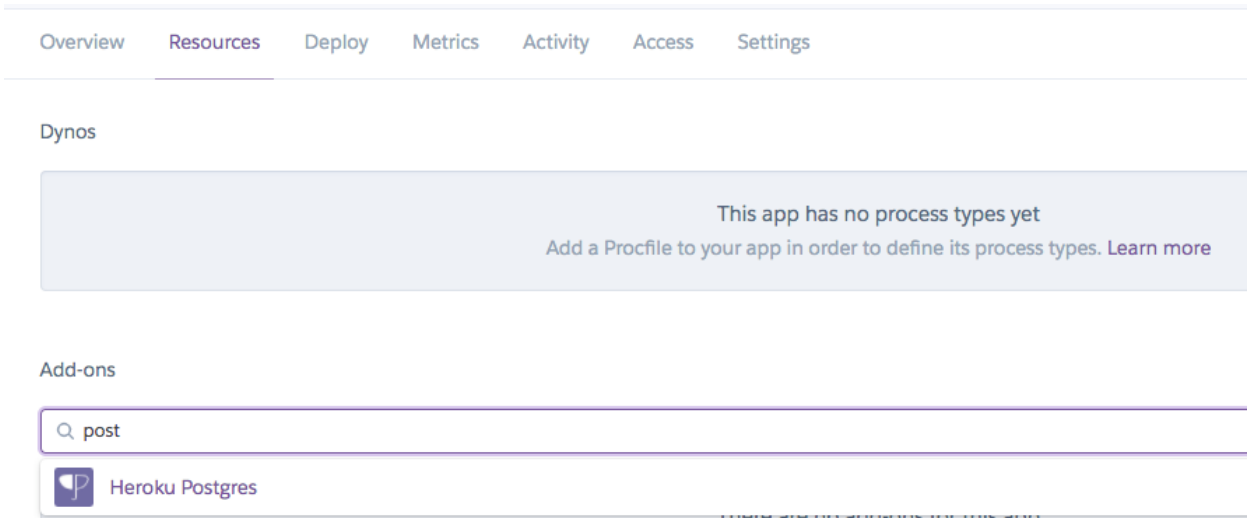
Part 2: Creating the Heroku App

- On Heroku, go to the `Deploy` section of your app's homepage, and follow the steps to deploy the app.

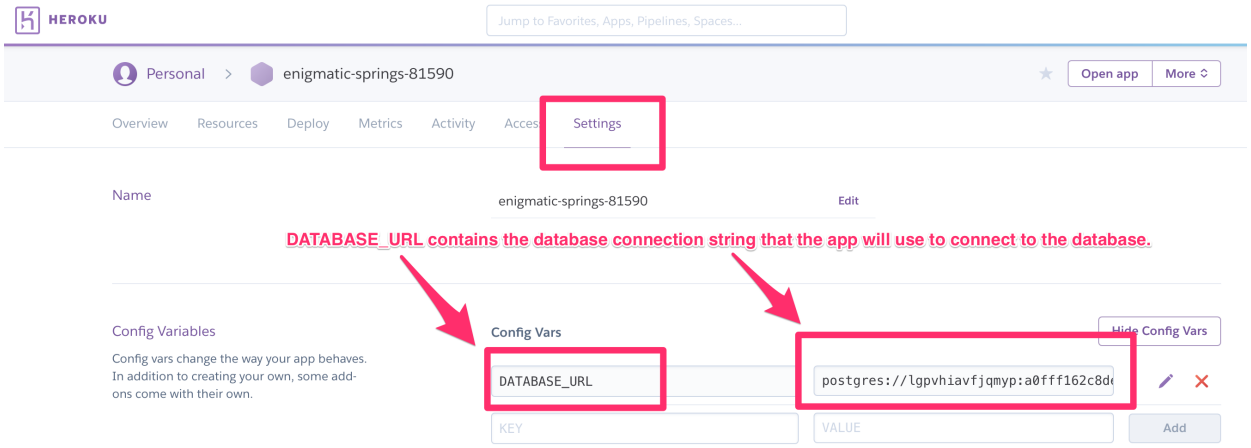


Part 3: Preparing the Database

- After creating a new app on Heroku, navigate to `Resources`:



  - Under `Add-ons`, add `Heroku Postgres`. Make sure to use the free version.
- Click on the add on, then navigate to settings and click on `Reveal Config Variables`.
- The connection string to the database should now be available:



- Heroku will automatically assign this URI string to the `DATABASE_URL` environment variable that is used within `app.py`. The code that is already in `app.py` will be able to use that environment variable to connect to the Heroku database.

```
# DATABASE_URL will contain the database connection string:
app.config['SQLALCHEMY_DATABASE_URI'] = os.environ.get('DATABASE_URL', '')
# Connects to the database using the app config
db = SQLAlchemy(app)
```

- After adding the database, the final step is to initialize the database. To do this, we use the heroku cli. From the terminal, type the following:

```
heroku run initdb.py
```

- Your database is now initialized, and you can open the application using `heroku open` from the terminal.

Copyright