Deploy Django On Fly.io

Install Fly

https://fly.io/docs/hands-on/install-flyctl/

Set up an account

https://fly.io/app/sign-up

Set Up Your Django App

Reconfigure Django Directory Structure

1. Rename the Django parent project folder something different
2. Move everything inside that folder into the top level of your repo
3. Delete the original (now empty) project folder

Install Packages

```
$ pip install gunicorn
```

```
$ pip install whitenoise
$ pip freeze > requirements.txt
```

Handle Static Pages In Django

In settings.py:
Add whitenoise middleware to the MIDDLEWARE list immediately after SecurityMiddleware

`"whitenoise.middleware.WhiteNoiseMiddleware",`

Then add:

```python
STATIC_ROOT = BASE_DIR / "staticfiles"

STORAGES = {
    'default': {
        'BACKEND': "django.core.files.storage.FileSystemStorage",
    },
    "staticfiles": {
        "BACKEND": "whitenoise.storage.CompressedManifestStaticFilesStorage",
    },
}
```

Setup Fly

```
$ fly launch --ha=false
```

Follow the instructions to connect your account. You should be fine with accepting the default settings they recommend.

This will create 2 files in your Django directory structure.

- fly.toml
- Dockerfile

SQLite

Our database is SQLite. It's just a file on disk. The fly machines don't preserve their disks between boots, so to preserve our database we need a “volume” that the database can live on.

```
$ fly volume create database_volume
```

Select the same region for the volume that Fly defaulted to when we created the app via “fly launch”.

Now we need to link the volume to the app. The app is running on a fly machine, we need to mount that volume as a directory on the fly machine.

In fly.toml: Add the following:

```
Unset
[[mounts]]
  source = 'database_volume'
  destination = '/mnt/volume_mount'
```

In fly.toml: overwrite ‘statics’:

```
Unset
[[statics]]
  guest_path = '/mnt/volume_mount/media'
  url_prefix = '/media/'
```

Now let's set the database to point to the file on disk. Most deployment systems (example: Fly, GitHub, Kubernetes) have a concept of 'secrets'. This is basically a value you can save securely and set into an environment variable. Let's save an environment variable that will hold the path to our database.

```
$ fly secrets set DATABASE_PATH=/mnt/volume_mount/production.sqlite
```

Update the Django settings.py to handle the database URL from the secret.
In settings.py - towards the top of the file: (you don’t have to rename APP_NAME - copy these just as they are)

```
Python
import os

APP_NAME = os.getenv("FLY_APP_NAME", None)

DATABASE_PATH = os.getenv("DATABASE_PATH", None)

CSRF_TRUSTED_ORIGINS = [f"https://{APP_NAME}.fly.dev"]
```

In Settings.py - overwrite ‘ALLOWED_HOSTS’ to allow local development and also production:

```
ALLOWED_HOSTS = ['127.0.0.1', f"{APP_NAME}.fly.dev"]
```

Add an environment variable to Fly to let us know that we are in production:

```
$ fly secrets set ENVIRONMENT='production'
```

In Settings.py - add 'ENVIRONMENT':

```
ENVIRONMENT = os.getenv('ENVIRONMENT', 'local')
```

In Settings.py - change 'DEBUG' to False if we are in anything other than a local environment:

```python
DEBUG = False
if ENVIRONMENT == 'local':
  DEBUG = True
```

In Settings.py - overwrite 'DATABASES' to use the DB url in our secret.

```python
DATABASES = {
	'default': {
		'ENGINE': 'django.db.backends.sqlite3',
		'NAME': DATABASE_PATH if APP_NAME else BASE_DIR / 'db.sqlite3',
	}
}
```

Secret Key

Move the secret key into the environment variables so that it isn't available to the public via our GitHub repository.

1. Type out a long and random string of characters.
2. `$ fly secrets set SECRET_KEY="your-long-and-random-string"`

In settings.py - change SECRET_KEY to:

```python
SECRET_KEY = os.getenv('SECRET_KEY', 'a default-value for local dev')
```

Handle CORS

$ `pip install django-cors-headers`

In settings.py

1. Add ‘corsheaders’ to INSTALLED_APPS
2. Add ‘corsheaders.middleware.CorsMiddleware’ to the top of the MIDDLEWARE list
3. Add these configurations:

```
JavaScript
CORS_ALLOWED_ORIGINS = [
       'http://localhost:8080',
       'http://the-url-of-your-react-app'
]

CORS_ALLOW_METHODS = [
    'GET',
    'POST',
    'PUT',
    'PATCH',
    'DELETE',
    'OPTIONS',
]

CORS_ALLOW_HEADERS = [
    'Content-Type',
    'Authorization',
]
```

Deploy!

Note - you will have to set up billing options before this will work.
Anytime you want to deploy your code to production, run this command:

$ `fly deploy`

$ `fly ssh console`

This logs you into the fly server so you can perform migrations:

$ `python manage.py migrate`

If you want to create a superuser for the admin backend:

$ `python manage.py createsuperuser`

To log back out of the fly server:

$ `exit`

FIRST DEPLOY:

$ `fly ssh console`

Collect static:

$ `python ./manage.py collectstatic`

Handle Image Uploads to Media Directory

In settings.py: Update the Media_Root so that it points to the volume if we are in production

```python
MEDIA_URL = '/media/'
MEDIA_ROOT = os.path.join(BASE_DIR, 'media')
if APP_NAME:
  MEDIA_ROOT = '/mnt/volume_mount/media/'
```

Connect React

In fly.io find your app and the app hostname. This is the url for your Django backend. Should look something like: https://the-name-of-the-app.fly.dev

Update any Axios calls to use this url.

Wait, it didn't work?!

On the off chance that the app doesn't start up for some reason, you can get your logs from:
$ `fly logs`

Resources

https://fly.io/docs/rails/advanced-guides/sqlite3/