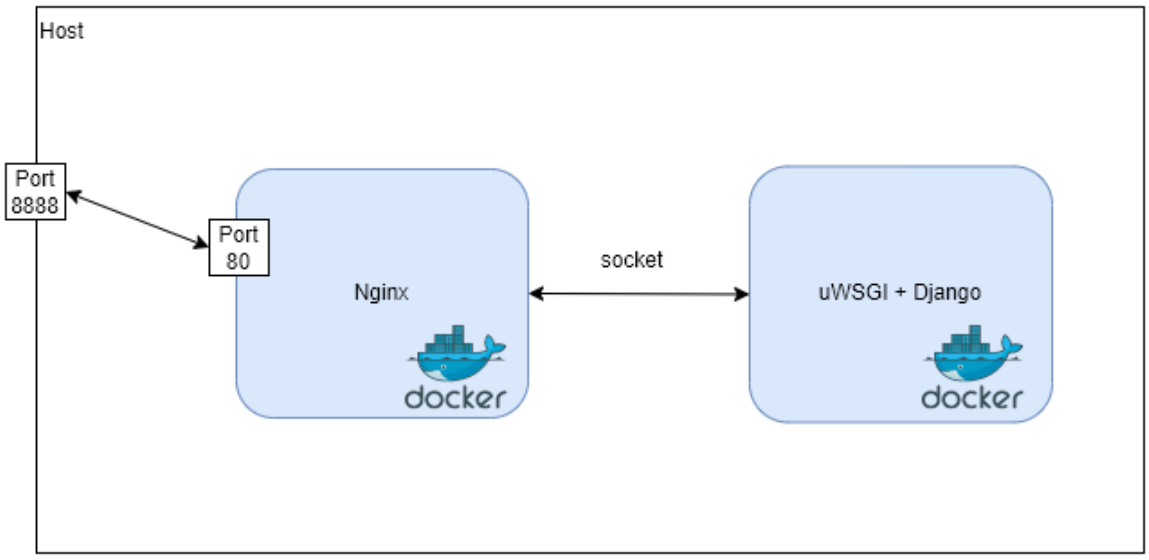
# How to Dockerize a Production-ready Django Application (Django + Nginx + uWSGI)

Imagine upgrading a system from one major version of Python to another. What are the challenges? I would say one of the biggest challenges is to reduce downtime during maintenance. Revert it as fast as we can if something goes wrong. Having been planning system upgrades on many projects with various teams, I have experienced many problems and frustrations caused by configuring and maintaining these "moving parts". I had experimented with different approaches to make the process a bit easier. My latest approach is to build the entire process around Docker technology. In my previous post, we built a live website by running the architecture of Django + Nginx + uWSGI. In this tutorial, we are going to dockerize the entire technical stacks.

The picture below illustrates what it would look like.

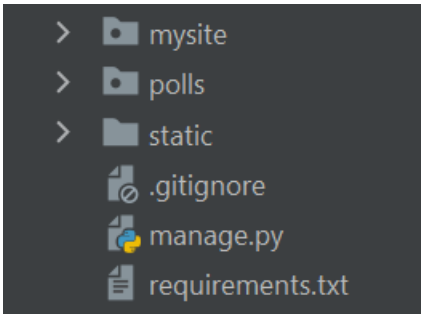



## Dependencies

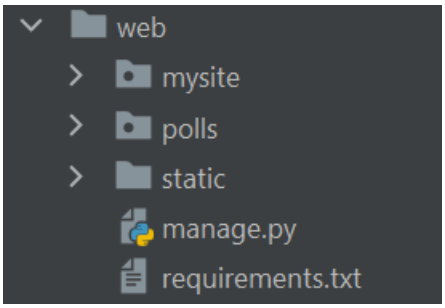
1. Docker and Docker Engine (To install them, see here)
2. Docker compose (To install it, see here)

## Step 1/5. Organize codebase

In the previous post, our codebase looks like the below.




Let's reorganize it. Create a **web** folder. Move over the entire codebase into it. We don't need **.gitignore** file. It should look like the below.




## Step 2/5. Dockerize codebase and uWSGI

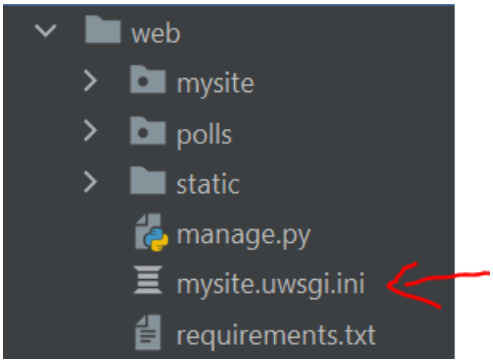
Create a **Dockerfile** file in the **web** folder. Its content is like the below.

```
FROM python:3.7-alpine
RUN mkdir /code
WORKDIR /code
COPY . /code

# uwsgi setup
RUN apk add python3-dev build-base linux-headers pcre-dev
RUN pip install uwsgi
RUN pip install -r requirements.txt


CMD ["uwsgi", "--ini", "/code/mysite.uwsgi.ini"]
```

Noticing that the **CMD** uses uWSGI to launch the Django application through a uWSGI config file, let's create it now. Create a **mysite.uwsgi.ini** file in the **web** folder. The content looks like the below.

```
[uwsgi]

socket = /tmp/uwsgi/mysite.sock
module = mysite.wsgi
master = true
processes = 2
chmod-socket = 666
vacuum = true
```

We are all set regarding developing a Docker container of codebase and uWSGI. The file structure should look like the one below at this point.




## Step 3/5. Dockerize Nginx

Create an **nginx** folder in the root folder. Then create an **nginx.conf** file. The main reason we need this file is to customize the Nginx setup in the container. The content looks like the below.

```
user  root;
worker_processes  auto;

error_log  /var/log/nginx/error.log notice;
pid        /var/run/nginx.pid;


events {
    worker_connections  1024;
}


http {
    include       /etc/nginx/mime.types;
    default_type  application/octet-stream;

    log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" "$http_x_forwarded_for"';

    access_log  /var/log/nginx/access.log  main;

    sendfile        on;
    #tcp_nopush     on;

    keepalive_timeout  65;

    #gzip  on;

    #include /etc/nginx/conf.d/*.conf;
    include /etc/nginx/sites-enabled/*;
}
```

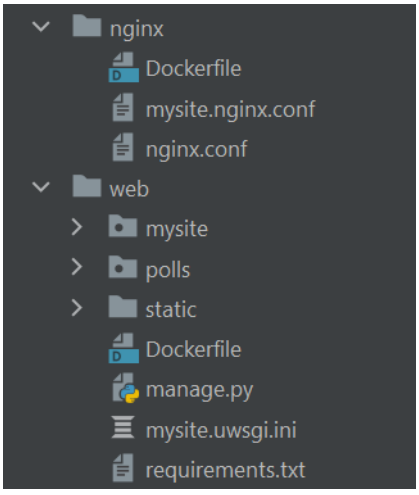
Create a **mysite.nginx.conf** file in the **nginx** folder. The content looks like the below.

```
upstream uwsgi {
    server unix:/tmp/uwsgi/mysite.sock;

}

server {
    listen      80;
    server_name 127.0.0.1;
    charset     utf-8;

    location /static {
        alias /var/www/mysite/assets;
    }

    location / {
        uwsgi_pass  uwsgi;
        include     /etc/nginx/uwsgi_params;
    }
}
```

Create a **Dockerfile** in the **nginx** folder. The content looks like the below.

```
FROM nginx:latest

COPY nginx.conf /etc/nginx/nginx.conf
COPY mysite.nginx.conf /etc/nginx/sites-available/mysite.nginx.conf
RUN mkdir /etc/nginx/sites-enabled
RUN ln -s /etc/nginx/sites-available/mysite.nginx.conf /etc/nginx/sites-enabled/

CMD ["nginx", "-g", "daemon off;"]
```

At this point, the Docker container of Nginx is all set. The file structure should look like the below.

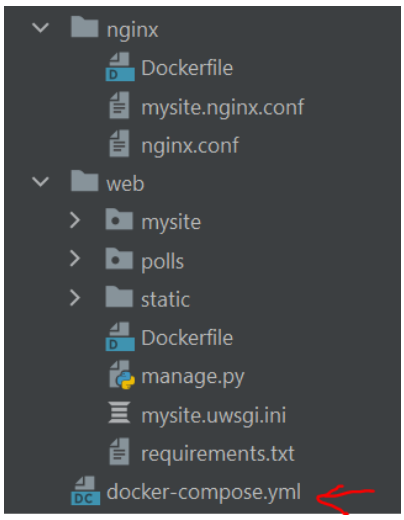


## Step 4/5. Develop Docker compose file

Create a **docker-compose.yml** in the root folder. The content looks like the below.

```
version: "3.9"
services:

 nginx:
  build: ./nginx/
  restart: always
  volumes:
   - uwsgi_data:/tmp/uwsgi/
   - web_static:/var/www/mysite/assets/:ro
  ports:
   - "8888:80"
  depends_on:
   - django

 django:
  build: ./web/
  restart: always
  command: >
   sh -c "python manage.py collectstatic --noinput
   && uwsgi --ini mysite.uwsgi.ini"
  volumes:
   - uwsgi_data:/tmp/uwsgi/
   - web_static:/code/static/
   - web_static:/var/www/mysite/assets/


volumes:
 uwsgi_data:
 web_static:
```

At this point, we are all set. The file structure should look like the below.



## Step 5/5. Fire up Docker container

Open Terminal/PowerShell. Go to the root folder of this project. Then type the following command.

docker compose build

We should see a lot of printings as below.

```
PS C:\Users\slow9\Desktop\Demo> docker compose build
[+] Building 26.7s (23/23) FINISHED
 => [demo_nginx internal] load build definition from Dockerfile                                                  0.0s
 => => transferring dockerfile: 32B                                                                              0.0s
 => [demo_django internal] load build definition from Dockerfile                                                 0.0s
 => => transferring dockerfile: 297B                                                                             0.0s
 => [demo_nginx internal] load .dockerignore                                                                     0.0s
 => => transferring context: 2B                                                                                  0.0s
 => [demo_django internal] load .dockerignore                                                                    0.0s
 => => transferring context: 2B                                                                                  0.0s
 => [demo_nginx internal] load metadata for docker.io/library/nginx:latest                                       1.7s
 => [demo_django internal] load metadata for docker.io/library/python:3.7-alpine                                 1.7s
 => [auth] library/python:pull token for registry-1.docker.io                                                    0.0s
 => [auth] library/nginx:pull token for registry-1.docker.io                                                     0.0s
 => [demo_django 1/7] FROM docker.io/library/python:3.7-alpine@sha256:1a1727003ad874410fc94d7c72b16c9ac082f4e0fcfe1da13d314a06bf33422b   2.6s
 => => resolve docker.io/library/python:3.7-alpine@sha256:1a1727003ad874410fc94d7c72b16c9ac082f4e0fcfe1da13d314a06bf33422b   0.0s
 => => sha256:1a1727003ad874410fc94d7c72b16c9ac082f4e0fcfe1da13d314a06bf33422b 1.65kB / 1.65kB                   0.0s
 => => sha256:c07b06c8726f780dd09fecaa6d4c4ef2a005835e4a5f47dad58d889798084ef4 1.37kB / 1.37kB                   0.0s
 => => sha256:710594a19ecc0a50080e800ddee523c51bec1be93b766ee1e387582084a13fbd 7.73kB / 7.73kB                   0.0s
 => => sha256:40e059520d199e1a1a259089077f2a0c879951c9a4540490bad3a0d7714c6ae7 2.81MB / 2.81MB                   0.3s
 => => sha256:4f950178bcecafaa0400ab3ea61cc27fd35a495600946da9fe805e2a46caa955 667.02kB / 667.02kB               0.1s
 => => sha256:bb5415fc276b9228fceecb3c886cc903b275ee341280196f22ca35b372ad125b 11.19MB / 11.19MB                 1.3s
 => => sha256:56303c2b76c4906aa6cd1ac179a2d771ddb452d9ef624a67c42f5828c9ff22c5 233B / 233B                       0.3s
 => => sha256:77c03d0e8ac9f440565c5cf4e6781a6037a4957de656a348c2d7f1172c8b0979 2.87MB / 2.87MB                   0.7s
 => => extracting sha256:40e059520d199e1a1a259089077f2a0c879951c9a4540490bad3a0d7714c6ae7                        0.2s
 => => extracting sha256:4f950178bcecafaa0400ab3ea61cc27fd35a495600946da9fe805e2a46caa955                        0.2s
 => => extracting sha256:bb5415fc276b9228fceecb3c886cc903b275ee341280196f22ca35b372ad125b                        0.6s
 => => extracting sha256:56303c2b76c4906aa6cd1ac179a2d771ddb452d9ef624a67c42f5828c9ff22c5                        0.0s
 => => extracting sha256:77c03d0e8ac9f440565c5cf4e6781a6037a4957de656a348c2d7f1172c8b0979                        0.3s
 => [demo_django internal] load build context                                                                    0.3s
 => => transferring context: 17.59MB                                                                             0.3s
 => [demo_nginx 1/5] FROM docker.io/library/nginx:latest@sha256:2275af0f20d71b293916f1958f8497f987b8d8fd8113df54635f2a5915002bf1   5.7s
 => => resolve docker.io/library/nginx:latest@sha256:2275af0f20d71b293916f1958f8497f987b8d8fd8113df54635f2a5915002bf1   0.0s
 => => sha256:83d487b625d8c7818044c04f1b48aabccd3f51c3341fc300926846bca0c439e6 1.57kB / 1.57kB                   0.0s
```

After the process is done, type the following command to make sure Docker images have been built successfully.

docker images

We should see two Docker images that are like the below.

```
PS C:\Users\slow9\Desktop\Demo> docker images
REPOSITORY    TAG       IMAGE ID       CREATED          SIZE
demo_django   latest    95a3c7c4f05a   9 minutes ago    400MB
demo_nginx    latest    dbc5205413b5   9 minutes ago    142MB
```

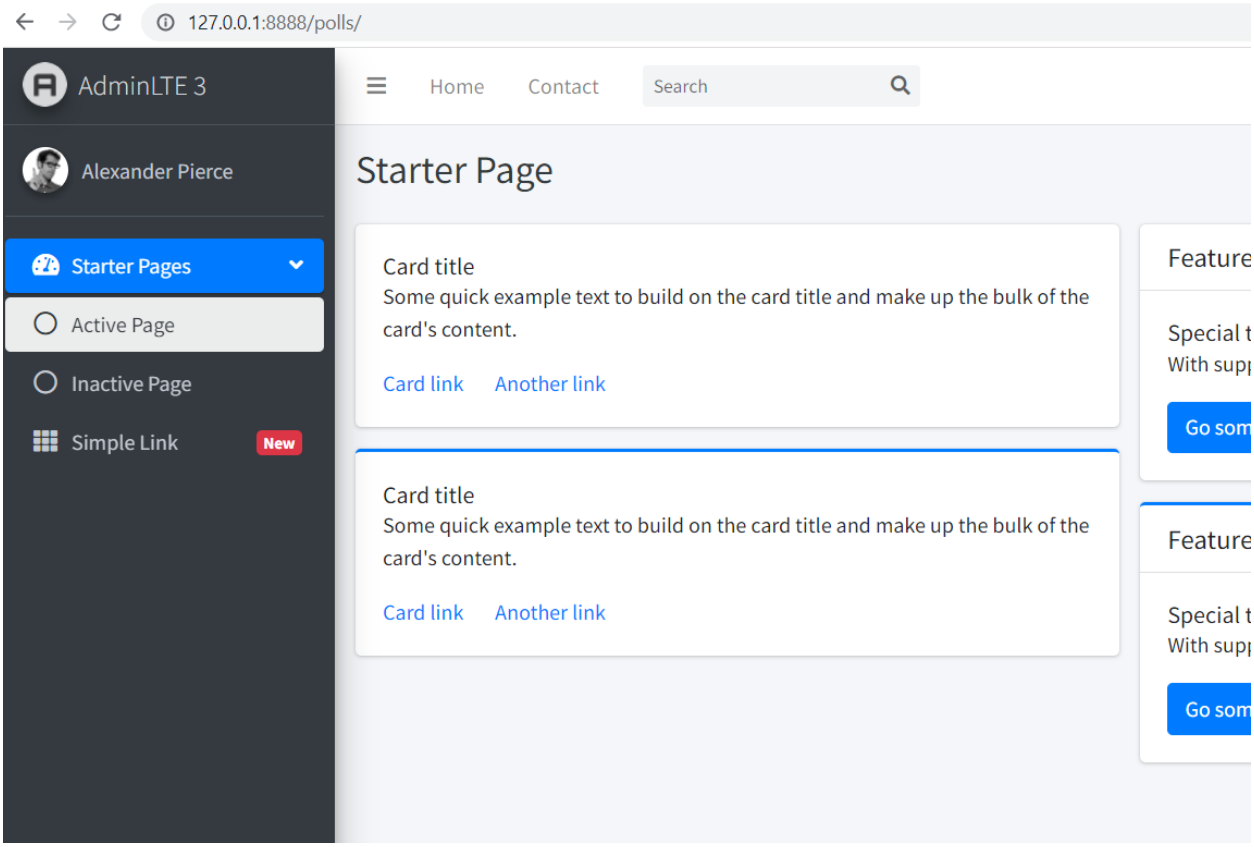
To fire up the container, type the following command.

docker compose up

We should see a lot of printings as below.

```
PS C:\Users\slow9\Desktop\Demo> docker compose up
[+] Running 5/4
 - Network demo_default      Created
 - Volume "demo_uwsgi_data"  Created
 - Volume "demo_web_static"  Created
 - Container demo-django-1   Created
 - Container demo-nginx-1    Created
Attaching to demo-django-1, demo-nginx-1
demo-django-1   |
demo-django-1   | 119 static files copied to '/var/www/mysite/assets', 110 unmodified.
demo-django-1   | [uWSGI] getting INI configuration from mysite.uwsgi.ini
demo-django-1   | *** Starting uWSGI 2.0.20 (64bit) on [Sun Apr  3 18:21:55 2022] ***
demo-django-1   | compiled with version: 10.3.1 20211027 on 03 April 2022 18:07:42
demo-django-1   | os: Linux-5.10.16.3-microsoft-standard-WSL2 #1 SMP Fri Apr 2 22:23:49 UTC 2021
demo-django-1   | nodename: a27d4efa8e33
demo-django-1   | machine: x86_64
demo-django-1   | clock source: unix
demo-django-1   | pcre jit disabled
demo-django-1   | detected number of CPU cores: 8
demo-django-1   | current working directory: /code
demo-django-1   | detected binary path: /usr/local/bin/uwsgi
demo-django-1   | uWSGI running as root, you can use --uid/--gid/--chroot options
demo-django-1   | *** WARNING: you are running uWSGI as root !!! (use the --uid flag) ***
demo-django-1   | your memory page size is 4096 bytes
demo-django-1   | detected max file descriptor number: 1048576
demo-django-1   | lock engine: pthread robust mutexes
demo-django-1   | thunder lock: disabled (you can enable it with --thunder-lock)
demo-django-1   | uwsgi socket 0 bound to UNIX address /tmp/uwsgi/mysite.sock fd 3
demo-django-1   | uWSGI running as root, you can use --uid/--gid/--chroot options
demo-django-1   | *** WARNING: you are running uWSGI as root !!! (use the --uid flag) ***
demo-django-1   | Python version: 3.7.13 (default, Mar 29 2022, 15:30:55)  [GCC 10.3.1 20211027]
demo-django-1   | *** Python threads support is disabled. You can enable it with --enable-threads ***
demo-django-1   | Python main interpreter initialized at 0x7efcb50492b0
demo-django-1   | uWSGI running as root, you can use --uid/--gid/--chroot options
demo-django-1   | *** WARNING: you are running uWSGI as root !!! (use the --uid flag) ***
demo-django-1   | your server socket listen backlog is limited to 100 connections
demo-django-1   | your mercy for graceful operations on workers is 60 seconds
demo-django-1   | mapped 218712 bytes (213 KB) for 2 cores
demo-django-1   | *** Operational MODE: preforking ***
demo-django-1   | WSGI app 0 (mountpoint='') ready in 0 seconds on interpreter 0x7efcb50492b0 pid: 1 (default app)
demo-django-1   | uWSGI running as root, you can use --uid/--gid/--chroot options
demo-django-1   | *** WARNING: you are running uWSGI as root !!! (use the --uid flag) ***
demo-django-1   | *** uWSGI is running in multiple interpreter mode ***
demo-django-1   | spawned uWSGI master process (pid: 1)
demo-django-1   | spawned uWSGI worker 1 (pid: 8, cores: 1)
demo-django-1   | spawned uWSGI worker 2 (pid: 9, cores: 1)
demo-nginx-1    | /docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
demo-nginx-1    | /docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
demo-nginx-1    | /docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
demo-nginx-1    | 10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
demo-nginx-1    | 10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf
demo-nginx-1    | /docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
demo-nginx-1    | /docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
demo-nginx-1    | /docker-entrypoint.sh: Configuration complete; ready for start up
```

Open browser. Type http://127.0.0.1:8888/polls/. You should see the page below. Congratulations!




## Conclusions

In this tutorial, we walked through the complete journey that starts from the original codebase to a container-based architecture. You may notice that it only takes a few seconds to fire up Docker containers. Having different versions of docker-compose.yml and docker images ready, we can confidently launch a system upgrade or revert it to the previous build “in the blink of an eye”. Hope this article gives you some ideas on addressing the problems of system deployment and upgrade. What do you think? Let me know your thoughts by leaving the comments. Thanks for reading.

I want to express my gratitude to Carlos Sandoval for introducing me to Docker technology.

I’ve uploaded the project on my Github. If you are interested in diving into it, please check it out here.

If you are interested in building this Django application, check out the full tutorial here.