**部署一个最简单的Django项目**

现在我们要在服务器上利用Docker部署下面一个最简单的Django项目。我们不使用uwsgi和nginx，数据库也使用默认的sqlite3，只把django放在一个容器里。整个项目结构如下所示，目前该项目放在宿主机(服务器)上。

mysite1

├── db.sqlite3

├── Dockerfile *# 用于生产docker镜像的Dockerfile*

├── manage.py

├── mysite1

│ ├── asgi.py

│ ├── \_\_init\_\_.py

│ ├── settings.py

│ ├── urls.py

│ └── wsgi.py

├── pip.conf *# 非必需。pypi源设置成国内，加速pip安装*

└── requirements.txt *# 项目只依赖Django，所以里面只有django==3.0.5一条*

注意：Django默认ALLOWED\_HOSTS = []为空，在正式部署前你需要修改settings.py, 把它设置为服务器实际对外IP地址，否则后面部署会出现错误，这个与docker无关。即使你不用docker部署，ALLOWED\_HOSTS也要设置好的。

**第一步：编写Dockerfile，内容如下：**

# 建立 python3.7 环境

FROM python:3.7

# 设置 python 环境变量

ENV PYTHONUNBUFFERED 1

# 在容器内/var/www/html/下创建 mysite1文件夹

RUN mkdir -p /var/www/html/mysite1

# 设置容器内工作目录

WORKDIR /var/www/html/mysite1

# 将当前目录文件加入到容器工作目录中（. 表示当前宿主机目录）

ADD . /var/www/html/mysite1

**第二步：使用当前目录的 Dockerfile 创建镜像，标签为 django\_docker\_img:v1。**

进入Dockerfile所在目录，输入如下命令：

# 根据Dockerfile创建名为django\_docker\_img的镜像，版本v1，.代表当前目录

sudo docker build -t django\_docker\_img:v1 .

# 查看镜像是否创建成功, 后面-a可以查看所有本地的镜像

sudo docker images

这是你应该可以看到有一个名为django\_docker\_img的docker镜像创建成功了，版本v1。

**第三步：根据镜像生成容器并运行，容器名为mysite1, 并将宿主机的80端口映射到容器的8000端口。**

*# 如成功，根据镜像创建mysite1容器并运行。宿主机80：容器8000。-d表示后台运行。*

sudo docker run -it -d --name mysite1 -p 80:8000 django\_docker\_img:v1

*# 查看容器状态，后面加-a可以查看所有容器列表，包括停止运行的容器*

sudo docker ps

*# 进入容器，如果复制命令的话，结尾千万不能有空格。*

sudo docker exec -it mysite1 /bin/bash

这时你应该可以看到mysite1容器开始运行了。使用sudo docker exec -it mysite1 /bin/bash即可进入容器内部。

**第四步：进入容器内部后，执行如下命令**

python3 manage.py makemigrations

python3 manage.py migrate

python3 manage.py runserver 0.0.0.0:8000

这时你打开Chrome浏览器输入[http://your\_server\_ip](https://link.zhihu.com/?target=http%3A//your_server_ip" \t "_blank)，你就可以看到你的Django网站已经上线了，恭喜你！

**从客户端到Docker容器内部到底发生了什么**

前面我们已经提到如果不指定容器网络，Docker创建运行每个容器时会为每个容器分配一个IP地址。我们可以通过如下命令查看。

sudo docker inspect mysite1 | grep "IPAddress"

用户访问的是宿主机服务器地址，并不是我们容器的IP地址，那么用户是如何获取容器内部内容的呢？答案就是端口映射。因为我们将宿主机的80端口(HTTP协议)隐射到了容器的8000端口，所以当用户访问服务器IP地址80端口时自动转发到了容器的8000端口。

注意：容器的IP地址很重要，以后要经常用到。同一宿主机上的不同容器之间可以通过容器的IP地址直接通信。一般容器的IP地址与容器名进行绑定，只要容器名不变，容器IP地址不变。

**把UWSGI加入Django容器中的准备工作**

在前面例子中我们使用了Django了自带的runserver命令启动了测试服务器，但实际生成环境中你应该需要使用支持高并发的uwsgi服务器来启动Django服务。尽管本节标题是把uwsgi加入到Django容器中，但本身这句话就是错的，因为我们Django的容器是根据django\_docker\_img:v1这个镜像生成的，我们的镜像里并没有包含uwsgi相关内容，只是把uwsgi.ini配置文件拷入到Django容器是不会工作的。

所以这里我们需要构建新的Dockerfile并构建新的镜像和容器。为了方便演示，我们创建了一个名为mysite2的项目，项目结构如下所示：

mysite2

├── db.sqlite3

├── Dockerfile *# 构建docker镜像所用到的文件*

├── manage.py

├── mysite2

│ ├── asgi.py

│ ├── \_\_init\_\_.py

│ ├── settings.py

│ ├── urls.py

│ └── wsgi.py

├── pip.conf

├── requirements.txt *# 两个依赖：django==3.0.5 uwsgi==2.0.18*

├── start.sh *# 进入容器后需要执行的命令，后面会用到*

└── uwsgi.ini *# uwsgi配置文件*

新的Dockerfile内容如下所示：

*# 建立 python3.7 环境*

FROM python:3.7

*# 镜像作者大江狗*

MAINTAINER DJG

*# 设置 python 环境变量*

ENV PYTHONUNBUFFERED 1

*# 设置pypi源头为国内源*

COPY pip.conf /root/.pip/pip.conf

*# 在容器内/var/www/html/下创建 mysite2 文件夹*

RUN mkdir -p /var/www/html/mysite2

*# 设置容器内工作目录*

WORKDIR /var/www/html/mysite2

*# 将当前目录文件拷贝一份到工作目录中（. 表示当前目录）*

ADD . /var/www/html/mysite2

*# 利用 pip 安装依赖*

RUN pip install -r requirements.txt

*# Windows环境下编写的start.sh每行命令结尾有多余的\r字符，需移除。*

RUN sed -i 's/\r//' ./start.sh

*# 设置start.sh文件可执行权限*

RUN chmod +x ./start.sh

start.sh脚本文件内容如下所示。最重要的是最后一句，使用uwsgi.ini配置文件启动Django服务。

*#!/bin/bash*

*# 从第一行到最后一行分别表示：*

*# 1. 生成数据库迁移文件*

*# 2. 根据数据库迁移文件来修改数据库*

*# 3. 用 uwsgi启动 django 服务, 不再使用python manage.py runserver*

python manage.py makemigrations**&&**

python manage.py migrate**&&**

uwsgi --ini /var/www/html/mysite2/uwsgi.ini

*# python manage.py runserver 0.0.0.0:8000*

uwsgi.ni配置文件内容如下所示。

**[**uwsgi**]**

project**=**mysite2

uid**=**www-data

gid**=**www-data

base**=**/var/www/html

chdir**=**%**(**base**)**/%**(**project**)**

module**=**%**(**project**)**.wsgi:application

master**=**True

processes**=**2

http**=**0.0.0.0:8000 *#这里直接使用uwsgi做web服务器，使用http。如果使用nginx，需要使用socket沟通。*

buffer-size**=**65536

pidfile**=**/tmp/%**(**project**)**-master.pid

vacuum**=**True

max-requests**=**5000

daemonize**=**/tmp/%**(**project**)**-uwsgi.log

*#设置一个请求的超时时间(秒)，如果一个请求超过了这个时间，则请求被丢弃*

harakiri**=**60

*#当一个请求被harakiri杀掉会，会输出一条日志*

harakiri-verbose**=**true

**单容器部署 Django + UWSGI**

**第一步：**生成名为django\_uwsgi\_img:v1的镜像

sudo docker build -t django\_uwsgi\_img:v1 .

**第二步**：启动并运行mysite2的容器

# 启动并运行mysite2的容器

sudo docker run -it -d --name mysite2 -p 80:8000 django\_uwsgi\_img:v1

**第三步：**进入mysite2的容器内部，并运行脚本命令start.sh

*# 进入容器，如果复制命令的话，结尾千万不能有空格。*

sudo docker exec -it mysite2 /bin/bash

*# 执行脚本命令*

sh start.sh

以上两句命令也可以合并成一条命令

sudo docker exec -it mysite2 /bin/bash start.sh

执行后效果如下所示。当你看到最后一句[uWSGI]时，说明uwsgi配置并启动完成。

这时你打开浏览器输入[http://your\_server\_ip](https://link.zhihu.com/?target=http%3A//your_server_ip)，你就可以看到你的Django网站已经上线了，恭喜你！这次是uwsgi启动的服务哦，因为你根本没输入python manage.py runserver命令。