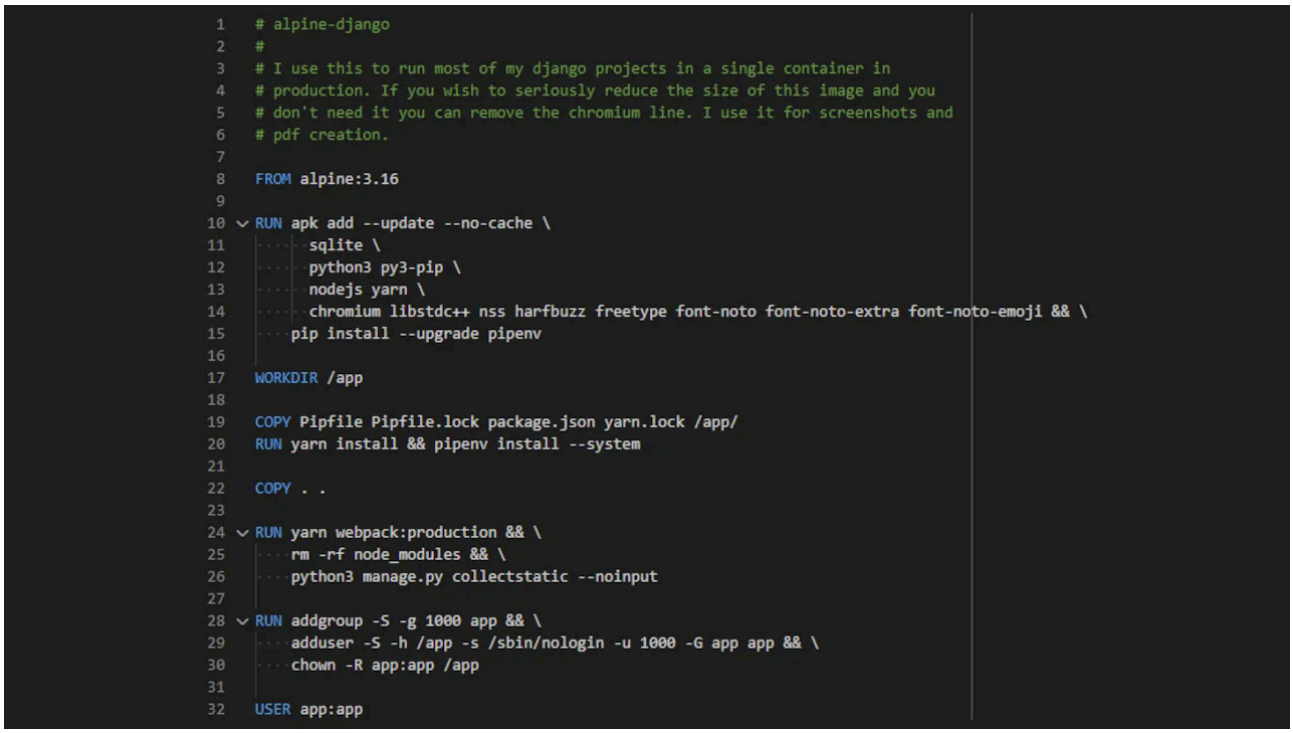
Running a simple Django website in Docker

Isaac Bythewood · 2022-05-14 · 3 min read

webdev



```
# alpine-django
#
# I use this to run most of my django projects in a single container in
# production. If you wish to seriously reduce the size of this image and you
# don't need it you can remove the chromium line. I use it for screenshots and
# pdf creation.

FROM alpine:3.16

RUN apk add --update --no-cache \
    sqlite \
    python3 py3-pip \
    nodejs yarn \
    chromium libstdc++ nss harfbuzz freetype font-noto font-noto-extra font-noto-emoji && \
  pip install --upgrade pipenv

WORKDIR /app

COPY Pipfile Pipfile.lock package.json yarn.lock /app/
RUN yarn install && pipenv install --system

COPY . .

RUN yarn webpack:production && \
  rm -rf node_modules && \
  python3 manage.py collectstatic --noinput

RUN addgroup -S -g 1000 app && \
  adduser -S -h /app -s /sbin/nologin -u 1000 -G app app && \
  chown -R app:app /app

USER app:app
```

Using Docker to run a simple production and development environments with a few extras thrown in. Easily customized to your preferred language or framework.

Docker is a near perfect solution for having project portability across a wide variety of host platforms. I use docker to run my websites on both development computers and servers. For Django my go to Dockerfile looks a little something like this.

```
# django
#
# I use this to run most of my django projects in a single container in
# production. If you wish to seriously reduce the size of this image and you
# don't need it you can remove the chromium line. I use it for screenshots and
# pdf creation.
#
# Make sure to change the below ENV variables to fit your needs and the gunicorn
# path to your applications asgi file.
#
# You can run this with:
#   docker build --tag overshard/django:latest .
#   docker run -d -p 80:8000 -e DJANGO_SETTINGS_MODULE=project.settings.production \
#     -v /srv/data:/data django:latest


FROM alpine:3.16

RUN apk add --update --no-cache \
      sqlite \
      python3 py3-pip \
```

```
      nodejs yarn \
      chromium libstdc++ nss harfbuzz freetype font-noto font-noto-extra font-noto-emoji &&
\
    pip install pipenv

COPY Pipfile Pipfile.lock package.json yarn.lock /app/

RUN yarn install && pipenv install --system

COPY . .

RUN yarn webpack:production && \
    rm -rf node_modules && \
    python3 manage.py collectstatic --noinput

RUN addgroup -S -g 1000 app && \
    adduser -S -h /app -s /sbin/nologin -u 1000 -G app app && \
    chown -R app:app /app

USER app:app

WORKDIR /app

VOLUME /data

EXPOSE 8000

ENV DJANGO_SETTINGS_MODULE=project.settings.production

CMD ["gunicorn", "project.asgi:application", "-k", "uvicorn.workers.UvicornWorker", "-w",
"4", "-b", ":8000", "--access-logfile", "-", "--error-logfile", "-"]
```

A few notes on my choices:

- This assumes you are just using an sqlite database but this can easily scale into using PostgreSQL in coordination with docker-compose
- I use webpack to build all of my static files on all of my sites hence why nodejs and yarn are included
- Chromium is used in most of my projects for generating PDFs and screenshots, I've found it the most reliable and consistent way of handling that functionality
- You'll need both gunicorn and uvicorn installed

You could remove Chromium and save ~400MB of space on a roughly ~450MB image if you have no use for it. It is by far the largest dependency here. I also often use docker-compose in conjunction with this Dockerfile.

```
# django
#
# I create a `.env` file in the same folder as my `Dockerfile` and
# `docker-compose.yml` file with the environmental variables below. Generally my
# server file struction is `/srv/git/app` for the git bare repository,
# `/srv/docker/app` for the git repo cloned from the bare repo, and
# `/srv/data/app` for the data directory mounted to the container.
#
# The ports are `8000:8000` because I often use Caddy or Nginx to reverse proxy
# to the container. You could probably just serve the app directly though with
# `8000:80` instead if you have no media files.

version: "3"
```

```
services:
  web:
    build: .
    volumes:
      - /srv/data/app/:/data/
    ports:
      - "8000:8000"
    command: gunicorn analytics.asgi:application -k uvicorn.workers.UvicornWorker -w 4 -
b :8000 --access-logfile - --error-logfile -
    restart: unless-stopped
    environment:
      DJANGO_SETTINGS_MODULE: ${DJANGO_SETTINGS_MODULE}
```

This can be used directly in production pretty well however I do put most of my websites behind Caddy using a reverse proxy. If you'd like to see my most up-to-date alpine-docker files you can check them out on my overshard/dockerfiles GitHub project.