# docker-django-channels-react-postgres-redis-nginx

A production-ready boilerplate that adds real-time support (WebSockets) to the classic “Django + React + Postgres + Nginx” stack.

| Layer | Tech | Notes |

|-------|------|-------|

| \*\*Backend\*\* | Django 4.2 LTS, Django Channels 4, Gunicorn 🎧 (UvicornWorker) | ASGI-first; WebSocket endpoint `/ws/echo/` |

| \*\*Frontend\*\* | React (DEV server in dev, Nginx-served build in prod) | Includes a tiny Echo demo in `src/App.js` |

| \*\*Realtime\*\* | Redis 7 (channels-redis) | One container per env, mounted volume for persistence |

| \*\*Database\*\* | Postgres 13-alpine | Standard setup |

| \*\*Proxy / Static\*\* | Nginx 1.25-alpine | Serves the React build and proxies `/api/\*\*` + WebSocket upgrade |

| \*\*Orchestration\*\* | Docker & Compose | No `version:` key (Compose v2+ best-practice) :contentReference[oaicite:0]{index=0} |

Two compose files are provided:

\* `docker-compose.yml` → \*\*development\*\*, hot-reload, four containers

\* `docker-compose.prod.yml` → \*\*production\*\*, multi-stage images, static React build, Gunicorn-Uvicorn

---

## 1 . Development Environment

### Containers

backend → Django runserver (ASGI) on :8000

frontend → React dev-server with HMR on :3000

db → Postgres on :5432

redis → Redis on :6379

pgsql

Copy

Edit

(All wired together on the `dev` network) :contentReference[oaicite:1]{index=1}

### Quick start

```bash

# 1. copy env template

cp .env.sample .env.dev # add secrets as needed

# 2. build images

docker compose build

# 3. run

docker compose up -d

React dev: http://localhost:3000

Django admin: http://localhost:8000/api/admin/

WebSocket test page: open the React UI and click Send Ping (see /ws/echo/).

To shut everything down:

bash

Copy

Edit

docker compose down

Stand-alone runs (optional)

bash

Copy

Edit

# Front-end

cd frontend && npm install && npm start

# Back-end

cd backend && python -m venv env && . env/bin/activate

pip install -r requirements/local.txt

python manage.py migrate && python manage.py runserver

2 . Production Environment

Extra containers

backend-prod, frontend-prod, redis-prod, db-prod, all on the prod network.

Gunicorn (+ UvicornWorker) handles both HTTP and WebSocket traffic.

One-time setup

bash

Copy

Edit

cp .env.sample .env.prod # DEBUG=0, strong SECRET\_KEY, etc.

cp .env.sample .env.prod.db # POSTGRES\_\* for prod DB

# (optional) set REDIS\_URL if not using the default:

# REDIS\_URL=redis://redis-prod:6379/0

Build & run

bash

Copy

Edit

docker compose -f docker-compose.prod.yml build

docker compose -f docker-compose.prod.yml up -d

Site root (React build): http://localhost

Django admin: http://localhost/api/admin/

Stop and remove:

bash

Copy

Edit

docker compose -f docker-compose.prod.yml down

3 . Environment variables

Name Typical dev value Purpose

SECRET\_KEY foo Django secret

DEBUG 1 / 0 Toggle debug

DJANGO\_ALLOWED\_HOSTS localhost 127.0.0.1 Space-separated list

REDIS\_URL redis://redis:6379/0 (dev) / redis://redis-prod:6379/0 (prod) Channel layer

Database credentials live in .env.\* and .env.\*.db as before.

4 . Development notes

Compose v2 – version: keys were removed because they are deprecated

ASGI routing lives in backend/app/asgi.py and wires realtime.routing.websocket\_urlpatterns.

React demo (src/App.js) shows the minimal client; feel free to delete once you implement your own UI.

Dependencies bumped to Django 4.2 LTS, Channels 4, and compatible libs