Полное пошаговое руководство по контейнеризации Django-проекта с Gunicorn и Nginx

Цели урока

- Контейнеризировать Django-проект с помощью Docker.
- Подключить PostgreSQL как основную базу данных.
- Использовать Gunicorn как WSGI-сервер для продакшена.
- Настроить Nginx как внешний HTTP-сервер, проксирующий запросы в Gunicorn и отдающий статику.
- Разобраться, зачем нужны Gunicorn и Nginx, и почему `runserver` использовать нельзя.

Почему нельзя использовать `runserver` в продакшене

`python manage.py runserver`:

- предназначен **только для разработки**;
- запускается в однопоточном режиме;
- не обрабатывает сигналы корректно;
- не масштабируется;
- не устойчив к падениям и нагрузке;
- не умеет работать за прокси;
- не поддерживает продакшн-фичи вроде таймаутов, логирования, мониторинга.

Gunicorn — это производительный WSGI-сервер:

- запускает несколько воркеров (процессов), каждый обрабатывает свой поток;
- устойчив к сбоям: если один воркер падает — остальные продолжают работать;
- умеет логировать, масштабироваться, подстраиваться под ядра CPU;
- стандартный сервер для Python-приложений на продакшене.

Gunicorn = промежуточный слой между Django и «внешним миром». Он понимает WSGI-протокол и умеет обрабатывать HTTP-запросы с высокой скоростью.

Почему нужен Nginx поверх Gunicorn

Gunicorn умеет только отдавать HTML и JSON. Он **не отдает** CSS, JS, картинки, медиа и пр.

Nginx — это полноценный HTTP-сервер:

- **Обработка статических файлов**: Nginx работает в 10-100 раз быстрее, чем Django при отдаче статики.
- **Обратное проксирование**: Nginx принимает HTTP-запрос и перенаправляет (проксирует) его в Gunicorn.

- **Балансировка нагрузки**: можно настроить пул Gunicorn-воркеров и распределение трафика.
- **Безопасность**: ограничение методов, защита от атак.
- **SSL/TLS**: именно Nginx должен принимать HTTPS-трафик (а не Gunicorn).
- **Кэширование**: можно кэшировать статику, JSON, страницы и уменьшать нагрузку на backend.

Таким образом, Gunicorn и Nginx выполняют разные роли:

Компонент	Отвечает за
Django	Генерация HTML, логика, ORM
Gunicorn	Запуск Django-приложения как WSGI
Nginx	HTTP-сервер, раздача файлов, прокси, безопасность

Без Nginx ваш проект:

- будет отдавать статику через Gunicorn → тормоза;
- будет уязвим к падениям;
- не сможет обслуживать HTTPS;
- не сможет эффективно масштабироваться.

Структура проекта

```
.
├── docker-compose.yml
├── Dockerfile                # для web (Django)
├── nginx/
│   ├── nginx.conf            # конфигурация Nginx
│   └── Dockerfile            # билд для Nginx
└── ... (код Django)
```

---

docker-compose.yml

```yaml
version: "3.8"

services:
  db:
    image: postgres:13
    container_name: db
    env_file:
      - .env
    environment:
      - POSTGRES_DB=${PG_NAME}
      - POSTGRES_USER=${PG_USER}
      - POSTGRES_PASSWORD=${PG_PASSWORD}
    volumes:
      - postgres_data:/var/lib/postgresql/data
```

```
  web:
    image: itg_web:latest
    build: .
    container_name: web
    command: python manage.py runserver 0.0.0.0:8000
    ports:
      - "8000:8000"
    depends_on:
      - db
    env_file:
      - .env
    environment:
      - DOCKERIZED=1
    volumes:
      - .:/app
      - media_volume:/media

  nginx:
    build:
      context: ./nginx
    container_name: nginx
    depends_on:
      - web
    volumes:
      - media_volume:/media
    ports:
      - "80:80"

volumes:
  postgres_data:
  media_volume:
```

Dockerfile (для web)

```
FROM python:3.12-slim

WORKDIR /app

RUN apt-get update && apt-get install -y \
        netcat-openbsd gcc libpq-dev && apt-get clean

COPY requirements.txt .
RUN pip install --upgrade pip && pip install -r requirements.txt

COPY entrypoint.sh wait-for-db.sh ./
RUN chmod +x entrypoint.sh wait-for-db.sh

ENV DOCKERIZED=1 \
    PYTHONUNBUFFERED=1 \
```

```
    PYTHONDONTWRITEBYTECODE=1

COPY . .

RUN mkdir -p /static && python manage.py collectstatic --no-input

ENTRYPOINT ["./entrypoint.sh"]
```

nginx/nginx.conf — подробный разбор

```
events {}

http {
  include       /etc/nginx/mime.types;       # подключаем типы файлов
  default_type  application/octet-stream;    # тип по умолчанию

  upstream django {
    server web:8000;   # имя сервиса из docker-compose
  }

  server {
    listen 80 default_server;                # принимаем входящие запросы
    server_name _;                           # localhost

    location /static/ {
      alias /static/;                        # отдаём статику напрямую
      access_log off;
      expires 30d;                           # кэшируем на 30 дней
    }

    location /media/ {
      alias /media/;                         # медиа (загруженные пользователями)
      access_log off;
      expires 30d;
    }

    location / {
      proxy_pass         http://django;      # проксируем всё остальное в Gunicorn
      proxy_set_header   Host $host;         # сохраняем заголовки
      proxy_set_header   X-Real-IP $remote_addr;
      proxy_set_header   X-Forwarded-For $proxy_add_x_forwarded_for;
      proxy_set_header   X-Forwarded-Proto $scheme;
    }
  }
}
```

nginx/Dockerfile

```
# 1-й этап — достаём статику из образа Django
FROM itg_web:latest AS static-src

# 2-й этап — финальный образ Nginx
FROM nginx:1.27-alpine

COPY nginx.conf /etc/nginx/nginx.conf
COPY --from=static-src /static /static
```

Таким образом, мы собираем статику в образе `web`, а затем копируем её в Nginx-контейнер. Это устраняет необходимость в `static_volume`, делает билд повторяемым и чистым.

Что мы получаем

- Контейнеризированный стек: PostgreSQL + Gunicorn + Django + Nginx.
- Отделение логики от рендеринга: Nginx обрабатывает HTTP-запросы, Django занимается бизнес-логикой.
- Устойчивый к нагрузке и удобный для продакшена проект.
- Понимание, как собрать и разложить проект на слои.
- Знание, чем отличается Gunicorn от runserver, и зачем нужен Nginx поверх Django.

Как запускать

```
docker compose build
docker compose up -d
```

Открываем в браузере: http://localhost/

Проверка статики: http://localhost/static/news/css/main.css → 200 OK

В дальнейшем можно добавить HTTPS, балансировщик, кэширование, образы с python-slim, CI/CD и др.