Повний гайд: Створення та деплоймент Django-проєкту з Docker

Цей посібник проведе вас через процес контейнеризації повноцінного Django-проєкту, використовуючи багатосервісну архітектуру з Docker Compose. Ми створимо п'ять окремих контейнерів, які працюватимуть як єдине ціле.

1. Створення Django-проєкту та Dockerfile

Перший крок — це створення Django-додатка і підготовка його до роботи в контейнері.

Структура проєкту:

```
django_project/
├── .env
├── docker-compose.yml
├── nginx/
│   ├── Dockerfile
│   └── nginx.conf
├── web/
│   ├── Dockerfile
│   ├── manage.py
│   ├── requirements.txt
│   └── myapp/
│       ├── __init__.py
│       ├── settings.py
│       ├── urls.py
│       └── ...
└── production.yml
```

Файл web/requirements.txt
Цей файл містить усі Python-бібліотеки, необхідні для роботи нашого додатка.

```
Django>=4.2,<5.0
psycopg2-binary
gunicorn
```

Файл web/Dockerfile
Це інструкція для створення образу нашого Django-додатка.

- FROM python:3.9-slim: Ми починаємо з мінімального образу Python, що робить наш контейнер легким.
- WORKDIR /usr/src/app: Встановлюємо робочий каталог, де будуть розміщені наші файли.
- COPY requirements.txt ./: Копіюємо файл залежностей.
- RUN pip install --no-cache-dir -r requirements.txt: Встановлюємо залежності. Прапорець --no-cache-dir зменшує розмір образу, не зберігаючи кеш.
- COPY . .: Копіюємо весь код нашого Django-проєкту в контейнер.
- CMD ["gunicorn", "myapp.wsgi:application", "--bind", "0.0.0.0:8000"]: Визначаємо команду запуску. Gunicorn — це високопродуктивний WSGI-сервер, який використовується для запуску Django в продакшн-середовищі. Він слухає запити на порту 8000 з усіх доступних інтерфейсів.

2. Контейнер Nginx та Gunicorn

Для розгортання Django-проєкту в Docker нам потрібні два ключові компоненти: **Gunicorn** та **Nginx**. Кожен з них виконує свою унікальну роль, що дозволяє досягти високої продуктивності та надійності.

Gunicorn: WSGI-сервер

Gunicorn (Green Unicorn) — це WSGI-сервер (Web Server Gateway Interface). . Це стандарт, який дозволяє Python-додаткам спілкуватися з веб-серверами.

- **Функція:** Gunicorn бере на себе відповідальність за запуск вашого Django-додатка, управління його процесами та обробку динамічних запитів. Замість того, щоб використовувати вбудований, однопотоковий сервер розробки Django, Gunicorn створює кілька робочих процесів, які можуть обробляти багато запитів одночасно, роблячи ваш додаток швидшим і надійнішим у реальних умовах.
- **Як це працює:** Коли Nginx отримує запит, який потрібно обробити Django (наприклад, перехід на сторінку `/` або `/about`), він перенаправляє цей запит на Gunicorn. Gunicorn, у свою чергу, обробляє його за допомогою вашого коду Django.

Nginx: Зворотний проксі та роздача статичних файлів

Nginx — це потужний веб-сервер, який в нашому стеку виконує дві ключові ролі:

- **Зворотний проксі (Reverse Proxy):** Він приймає всі запити, що надходять з Інтернету, і перенаправляє їх на потрібні контейнери. Це як "диспетчер" для

вашого трафіку.

- **Роздача статичних файлів:** Nginx самостійно та ефективно роздає статичні файли (CSS, JavaScript, зображення), що значно зменшує навантаження на Django та Gunicorn. Це дозволяє Django зосередитись лише на бізнес-логіці.

Файл `nginx/Dockerfile` Це інструкція для створення образу Nginx.

```
# Використання офіційного образу Nginx
FROM nginx:alpine

# Копіювання нашого власного конфігураційного файлу
COPY ./nginx.conf /etc/nginx/conf.d/default.conf

# Відкриття порту 80
EXPOSE 80
```

Файл `nginx/nginx.conf` Це конфігурація для Nginx.

```
server {
    listen 80;
    server_name localhost;

    location / {
        proxy_pass http://web:8000;
        proxy_set_header Host $host;
        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;
    }

    location /static/ {
        alias /usr/src/app/static/;
    }
}
```

`listen 80`: Nginx слухає запити на стандартному HTTP-порті 80.

`location /`: Усі запити, що не відповідають іншим `location`, перенаправляються на

наш Django-контейнер (`web`), який працює на порту `8000`.

`location /static/`: Запити, що починаються з `/static/`, Nginx обробляє сам, віддаючи файли з каталогу `/usr/src/app/static/`.

3. Налаштування контейнера з Postgres

PostgreSQL — це наша база даних. Використання окремого контейнера забезпечує ізоляцію та зручність управління.

Файл .env
Цей файл містить змінні середовища, які будуть доступні для всіх сервісів. Замість того, щоб вказувати паролі безпосередньо у docker-compose.yml, ми використовуємо цей файл.

```
POSTGRES_DB=django_db
POSTGRES_USER=admin
POSTGRES_PASSWORD=password
```

Сервіс postgres у docker-compose.yml

```
  postgres:
    restart: always
    image: postgres:15
    volumes:
      - pgdata:/var/lib/postgresql/data/
    env_file: .env
```

- image: postgres:15: Ми використовуємо офіційний образ PostgreSQL версії 15.
- volumes: Монтуємо **том** pgdata до каталогу, де Postgres зберігає дані. Це гарантує, що ваші дані збережуться, навіть якщо контейнер буде видалено.
- env_file: Підключаємо файл .env, щоб отримати змінні для налаштування бази даних.

4. Налаштування Redis

Redis — це in-memory база даних. У нашому стеку вона може використовуватися для:

- **Кешування:** Швидке зберігання тимчасових даних для прискорення роботи додатка.
- **Черги завдань:** Обробка асинхронних завдань, наприклад, відправка електронних листів.

Сервіс redis у docker-compose.yml

```
  redis:
    restart: always
    image: redis:latest
    volumes:
      - redisdata:/data
```

- image: redis:latest: Використовуємо офіційний образ Redis.
- volumes: Зберігаємо дані Redis в окремому томі redisdata.

5. Налаштування томів (Volumes)

Томи (Volumes) — це механізм Docker для зберігання даних, які повинні залишатися постійними. На відміну від даних, що зберігаються всередині контейнера, дані в томах не зникають, коли контейнер видаляється.

Блок volumes у docker-compose.yml

```
volumes:
  web-static:
  pgdata:
  redisdata:
```

- web-static: Використовується для зберігання статичних файлів, зібраних Django, щоб Nginx міг їх роздавати.
- pgdata: Зберігає дані нашої бази даних PostgreSQL.
- redisdata: Зберігає дані Redis.

6. Фінальний файл docker-compose.yml

Ось як виглядає повний файл, який об'єднує всі наші сервіси.

```yaml
version: '3.8'

services:
  web:
    restart: always
    build: ./web
    expose:
      - "8000"
    volumes:
      - web-static:/usr/src/app/static
    env_file: .env
    depends_on:
      - postgres
      - redis

  nginx:
    restart: always
    build: ./nginx
    ports:
      - "80:80"
    volumes:
      - web-static:/usr/src/app/static
    depends_on:
      - web

  postgres:
    restart: always
    image: postgres:15
    volumes:
      - pgdata:/var/lib/postgresql/data
    env_file: .env

  redis:
    restart: always
    image: redis:latest
    volumes:
      - redisdata:/data

volumes:
  web-static:
  pgdata:
  redisdata:
```

7. Локальний запуск та деплоймент

Налаштування Docker Machine

1. Створіть нову машину для Docker:
$ docker-machine create -d virtualbox dev

2. Встановіть машину dev як активну:
$ eval $(docker-machine env dev)

3. Перевірте, які машини запущені:
$ docker-machine ls

Локальний запуск

1. У кореневому каталозі проєкту (django_project/) створіть файл .env.
2. Виконайте команду для запуску всіх контейнерів:

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

3. Після запуску, виконайте міграції Django та зберіть статичні файли:

```
$ docker-compose run --rm web python myapp/manage.py migrate
$ docker-compose run --rm web python myapp/manage.py collectstatic
--noinput
```

4. Ваш додаток доступний за адресою http://localhost.

Детальний гайд: Деплоймент на AWS EC2

Для деплойменту ми будемо використовувати віртуальний сервер в хмарі — **AWS EC2**. Цей процес складається з кількох ключових етапів.

Крок 1: Підготовка інстансу EC2

1. **Зайдіть в консоль AWS** і перейдіть до сервісу EC2.
2. Натисніть **"Launch instance"** (Запустити інстанс).
3. **Виберіть операційну систему**. Рекомендується використовувати Ubuntu Server (версія 20.04 LTS або новіша), оскільки вона добре підтримує Docker.
4. **Виберіть тип інстансу**. Для тестування підійде t2.micro (він входить до Free Tier).

5. **Створіть Key Pair (пару ключів)**. Це дуже важливо! Надайте ключу ім'я (наприклад, django-key) і натисніть "Create key pair". Ваш приватний ключ (django-key.pem) буде автоматично завантажено. **Збережіть цей файл у безпечному місці, він потрібен для SSH-підключення.**
6. **Налаштуйте мережеві правила (Security Group)**. Натисніть "Edit" і створіть нову групу. Переконайтеся, що в ній дозволений вхідний трафік для таких портів:
   - **Port 22 (SSH)**: Для підключення до сервера.
   - **Port 80 (HTTP)**: Щоб ваш веб-додаток був доступний через браузер.
7. **Запустіть інстанс**. Після запуску, ви зможете знайти його публічну IP-адресу в інформації про інстанс.

Крок 2: Підключення до інстансу через SSH

1. **Відкрийте термінал** на вашій локальній машині.
2. **Змініть дозволи** на ваш приватний ключ .pem, щоб він був доступний тільки для вас:

```
$ chmod 400 path/to/your-key.pem
```

3. **Підключіться до інстансу** за допомогою команди SSH. Замініть path/to/your-key.pem на шлях до вашого ключа, а your-ec2-public-ip на публічну IP-адресу вашого інстансу.

```
$ ssh -i path/to/your-key.pem ubuntu@your-ec2-public-ip
```

   - ubuntu — це стандартний користувач для інстансів Ubuntu.

Крок 3: Встановлення Docker та Docker Compose

1. Після успішного підключення, **оновіть пакети** на сервері:

```
$ sudo apt-get update
```

2. **Встановіть Docker та Docker Compose**:

```
$ sudo apt-get install -y docker.io docker-compose
```

3. **Додайте користувача до групи Docker**, щоб ви могли виконувати команди Docker

без sudo:

```
$ sudo usermod -aG docker $USER
```

4. **Щоб зміни вступили в силу**, вийдіть з SSH-сесії і залогіньтесь знову.

Крок 4: Копіювання проєкту на інстанс

1. На вашій **локальній машині**, у кореневому каталозі проєкту, скопіюйте всі файли на EC2-інстанс за допомогою команди scp:

```
$ scp -r -i path/to/your-key.pem django_project/
ubuntu@your-ec2-public-ip:~
```

   - scp (secure copy) — безпечно копіює файли через SSH.
   - -r — копіює каталоги рекурсивно.
   - ~ — вказує на домашній каталог користувача на сервері.

Крок 5: Запуск проєкту на EC2

1. Знову підключіться до EC2 через SSH.
2. Перейдіть до каталогу проєкту:

```
$ cd django_project
```

3. Запустіть Docker-стек. Тепер усі ці команди виконуються на віддаленому сервері:

```
$ docker-compose up --build -d
$ docker-compose run --rm web python myapp/manage.py migrate
$ docker-compose run --rm web python myapp/manage.py collectstatic
--noinput
```

Крок 6: Доступ до додатка

- Відкрийте публічну IP-адресу вашого EC2-інстансу в браузері, і ви побачите свій Django-додаток.
- Якщо щось не працює, перевірте, чи коректно налаштовані правила в Security Group.