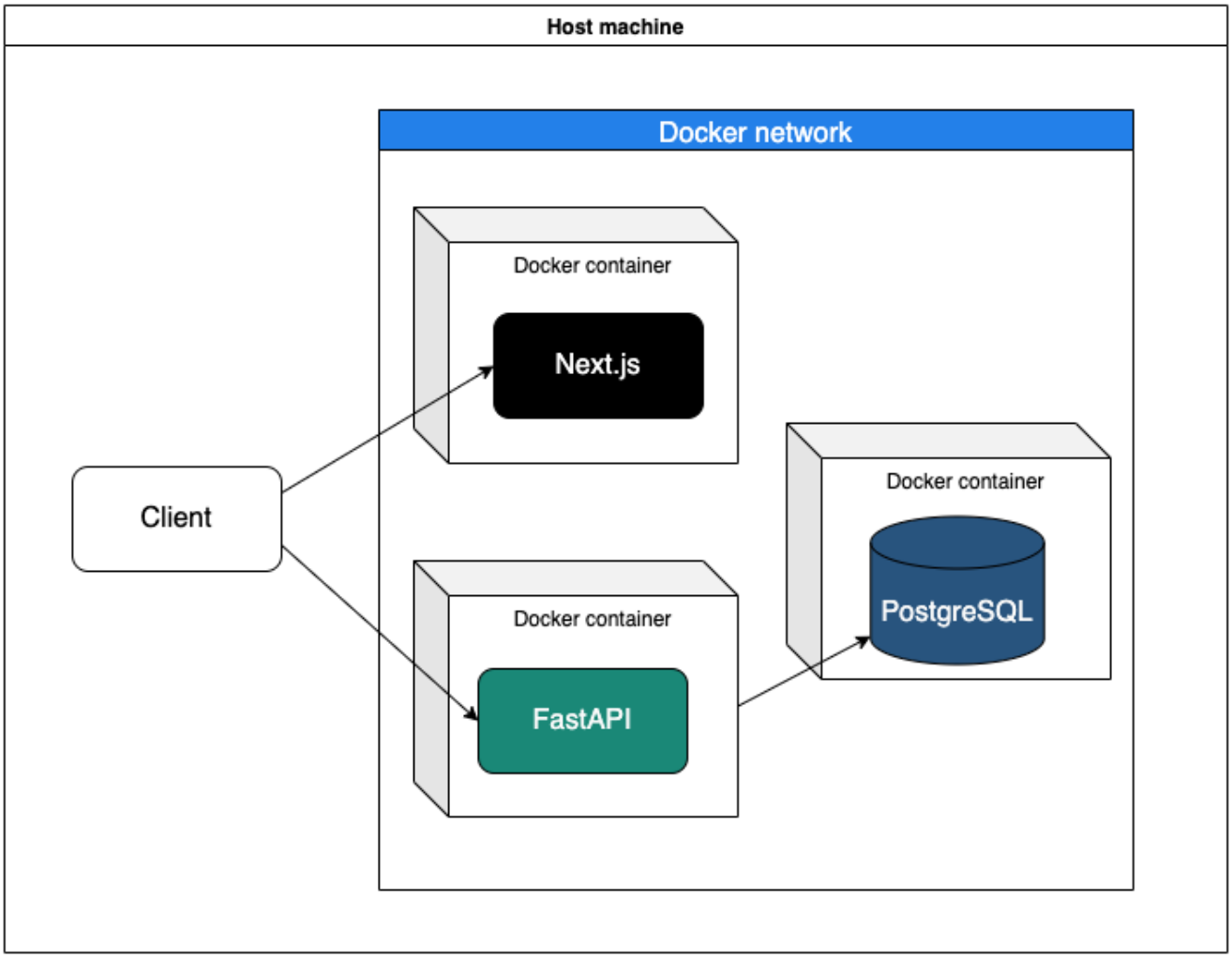
Как разработать полный стек Next.js Приложение FastAPI, PostgreSQL с использованием Docker

12.12.2021

Это продолжение предыдущих руководств по сборке и развертыванию full-stack Next.js Приложение FastAPI и PostgreSQL.



Ветка для этого руководства:

https://github.com/travisluong/nfp-boilerplate/tree/tutorial-3-how-to-develop-using-docker

Полный проект:

https://github.com/travisluong/nfp-boilerplate

Docker

Docker - это инструмент, который позволяет запускать ваше программное обеспечение в контейнерах. Контейнер - это процесс и файловая система, которые изолированы от процессов и файловой системы вашего хост-компьютера. Пользовательская файловая система предоставляется образом контейнера. Используя docker, вы сможете каждый раз запускать свое программное обеспечение в согласованной среде. Это особенно полезно, когда вам приходится подключать к проекту множество разработчиков и в итоге вы оказываетесь в ситуации, когда он работает на одной машине, но не работает на другой из-за различий в хост-машине. Docker устраняет эту очень распространенную проблему.

Настройка

Первым шагом является установка Docker Desktop. https://www.docker.com

Как только вы скачали и установили его, вы готовы к следующему шагу.

Откройте `nfp-boilerplate` проект в VSCode или любом другом текстовом редакторе, который вы используете.

Создание и запуск контейнеров

Изображения, используемые в этом руководстве, можно найти на Docker Hub. Мы используем официальные образы python и node.

Серверная часть

Создайте `Dockerfile` в `nfp-backend`.

```
FROM python:3.9

WORKDIR /usr/src/app

COPY requirements.txt ./
RUN pip install --no-cache-dir -r requirements.txt

COPY . .

CMD [ "uvicorn", "main:app", "--reload", "--host", "0.0.0.0" ]
```

В `.env` вы захотите изменить `localhost` строку в вашем `DATABASE_URL` на `host.docker.internal`. Это должно выглядеть примерно так.

```
DATABASE_URL=postgresql://nfp_boilerplate_user:password@host.docker.internal/nfp_boi
lerplate_dev
```

Это позволяет приложению внутри контейнера подключаться к экземпляру PostgreSQL, запущенному на вашем хост-компьютере.

Создайте образ.

```
$ docker build -t nfp-backend .
```

`-t` Флаг помечает изображения.

Запустите контейнер.

```
$ docker run -p 8000:8000 -it --rm --name nfp-backend-running nfp-backend
```

- `-p` Флаг сопоставляет порт хоста с портом контейнера. `HOST:CONTAINER`
- Флаги `-i` и `-t` обычно используются в комбинации для доступа к терминалу. Оно сокращено до `-it`.
- `--rm` Флаг гарантирует автоматическое удаление контейнера, когда он существует.
- `nfp-backend-running` это имя запущенного контейнера.
- `nfp-backend` это название изображения.

Интерфейс

Создайте `Dockerfile` в `nfp-frontend`.

```
FROM node:16

WORKDIR /usr/src/app

COPY package.json ./
RUN npm install

COPY . .

CMD [ "npm", "run", "dev" ]
```

Вставьте компакт-диск в `nfp-frontend` и запустите сборку:

```
$ docker build -t nfp-frontend .
```

Запустите контейнер:

```
$ docker run -p 3000:3000 -it --rm --name nfp-frontend-running nfp-frontend
```

Привязка монтирует

Вы узнали, как создавать и запускать контейнер, однако это не идеальный способ разработки, поскольку изменения в ваших файлах в файловой системе вашего хоста не будут отражены в контейнере. Чтобы устранить эту проблему, вы можете использовать привязку.

```
$ docker run -p 8000:8000 -it --rm --name nfp-backend-running -v "$PWD":/usr/src/app
nfp-backend
```

`-v "$PWD":/usr/src/app` Ваш текущий каталог будет сопоставлен с `/usr/src/app` каталогом в контейнере. Теперь внесенные вами изменения будут отражены в контейнере.

Для интерфейса вам нужно будет создать этот `.babelrc` файл:

```
{
    "presets": ["next/babel"]
}
```

В противном случае вы получите эту ошибку: `error - Failed to load SWC binary, see more info here: https://nextjs.org/docs/messages/failed-loading-swc`

Теперь привязку mount можно использовать для интерфейса:

```
$ docker run -p 3000:3000 -it --rm --name nfp-frontend-running -v
"$PWD":/usr/src/app nfp-frontend
```

Docker Compose

Эти команды docker могут быть довольно длинными и громоздкими для ввода. Docker Compose позволяет записывать все конфигурации в файл YAML и запускать все одной командой.

Создайте `docker-compose.yml` в корне проекта.

```
version: "3.9"

services:
  backend:
    build: nfp-backend
    ports:
      - 8000:8000
    volumes:
      - ./nfp-backend:/usr/src/app
  frontend:
    build: nfp-frontend
    ports:
      - 3000:3000
    volumes:
      - ./nfp-frontend:/usr/src/app
```

Запустите docker compose:

```
$ docker compose up
```

С помощью одной команды вы можете создавать и запускать все контейнеры одновременно.

Контейнер базы данных

Давайте добавим контейнер Postgres в нашу настройку docker. Добавьте следующее в `docker-compose.yml` раздел `services`.

```
  db:
    image: postgres:14
    restart: always
    environment:
      POSTGRES_USER: nfp_boilerplate_user
      POSTGRES_DB: nfp_boilerplate_dev
      POSTGRES_PASSWORD: password
```

Поскольку мы переходим от использования экземпляра базы данных, работающего на нашем хосте, к экземпляру, работающему внутри контейнера, нам придется изменить наш хост базы данных в `.env` и `alembic.ini`.

В `.env`:

```
DATABASE_URL=postgresql://nfp_boilerplate_user:password@db/nfp_boilerplate_dev
```

В `alembic.ini`:

```
sqlalchemy.url = postgresql://nfp_boilerplate_user:password@db/nfp_boilerplate_dev
```

Обратите внимание, что хост изменился на `db` в обоих файлах.

Введите `ctrl+c`, чтобы завершить текущий процесс.

Затем снова запустите docker compose:

```
$ docker compose up
```

Нам нужно будет снова выполнить миграцию, поскольку это новая база данных.

Откройте терминал в серверном контейнере. Во-первых, начните с проверки запущенных процессов docker.

```
$ docker ps
```

Затем запустите это:

```
$ docker exec -it nfp-boilerplate_backend_1 bash
```

Примечание: Убедитесь, что имя контейнера соответствует тому, что было указано в `docker ps`.

Внутри контейнера запустите миграции:

```
$ alembic upgrade head
```

Затем выйдите из контейнера:

```
$ exit
```

Откройте терминал в контейнере базы данных:

```
$ docker exec -it nfp-boilerplate_db_1 bash
```

Откройте сеанс psql:

```
$ psql -U nfp_boilerplate_user nfp_boilerplate_dev
```

Убедитесь, что таблицы были созданы.

```
# \dt
```

Убедитесь, что приложение все еще функционирует в, перейдя к `http://localhost:3000/notes` в вашем браузере.

Сети Docker

Стоит упомянуть, что Docker Compose управляет всей настройкой сети Docker. Все службы в вашем файле docker-compose смогут взаимодействовать друг с другом с помощью своего имени хоста, которое автоматически устанавливается в качестве имени их службы. Например `backend`, `frontend` и `db`.

Например, внутри контейнера внешнего интерфейса вы можете вызывать серверную часть и наоборот:

```
$ curl backend:8000
```

Вот почему мы смогли изменить хост базы данных на `db` в настройках базы данных.

Именованные тома

Если вы хотите сохранить данные своей базы данных, вы можете использовать для этого именованный том.

Добавьте следующее в `docker-compose.yml`:

```
    volumes:
      - pgdata:/var/lib/postgresql/data

volumes:
  pgdata:
```

Финал `docker-compose.yml` должен выглядеть следующим образом:

```
version: "3.9"

services:
  backend:
```

```
    build: nfp-backend
    ports:
      - 8000:8000
    volumes:
      - ./nfp-backend:/usr/src/app
  frontend:
    build: nfp-frontend
    ports:
      - 3000:3000
    volumes:
      - ./nfp-frontend:/usr/src/app
  db:
    image: postgres:14
    restart: always
    environment:
      POSTGRES_USER: nfp_boilerplate_user
      POSTGRES_DB: nfp_boilerplate_dev
      POSTGRES_PASSWORD: password
    volumes:
      - pgdata:/var/lib/postgresql/data

volumes:
  pgdata:
```

Введите `ctrl+c`, чтобы завершить процесс создания docker.

Затем запустите:

```
$ docker compose down
```

Затем верните контейнеры обратно с помощью:

```
$ docker compose up
```

Данные в контейнере должны были сохраниться. Без именованного тома данные были бы удалены навсегда, если изображение будет удалено.

Чтобы увидеть ваши тома:

```
$ docker volume ls
```

Заключение

Поздравляю. Вы настроили согласованную среду разработки с помощью Docker. Если вы часто сталкиваетесь с проблемой “Это работает на моей машине”, тогда рассмотрите возможность

использования Docker. Небольшие инвестиции сейчас могут сэкономить вам и вашей команде потраченные впустую часы в будущем.

Предыдущий: Как развернуть Next.js FastAPI и PostgreSQL с использованием сценариев оболочки
Далее: Как построить поток аутентификации пользователя с Next.js FastAPI и PostgreSQL
