

¡Magnífica decisión! Pasar a **Docker** es el momento en que tu proyecto deja de ser un "script local" y se convierte en una **Arquitectura de Software** transportable.

Con esta configuración, encapsularemos el **Nexo Sinérgico** en tres contenedores aislados que orquestaremos juntos. Esto garantiza que si funciona en tu máquina, funcionará igual en AWS, Azure o Google Cloud.

Aquí tienes la **Arquitectura de Contenedores** que vamos a construir:

---

## Paso 1: Estructura de Carpetas

Para que Docker funcione limpiamente, necesitamos organizar tu código. Asegúrate de que tu proyecto tenga esta estructura (si no la tiene, mueve los archivos ahora):

```
Plaintext  
/nexo-sinergico-project  
|  
|   └── /backend      # Tu código Python/FastAPI actual  
|       ├── main.py  
|       ├── requirements.txt  
|       └── Dockerfile  # (Nuevo archivo)  
  
|   └── /frontend      # Tu código React/Next.js actual  
|       ├── package.json  
|       └── /src  
|           └── Dockerfile  # (Nuevo archivo)  
  
└── docker-compose.yml # (El Director de Orquesta)
```

---

## Paso 2: Los **Dockerfiles** (Instrucciones de Construcción)

Estos archivos le dicen a Docker cómo construir cada pieza del sistema.

### A. El Backend ([/backend/Dockerfile](#))

Creamos un entorno Python ligero y limpio.

Dockerfile

```
# Usamos una imagen oficial de Python ligera  
FROM python:3.10-slim
```

```
# Evita que Python genere archivos .pyc y buffer de salida
ENV PYTHONDONTWRITEBYTECODE 1
ENV PYTHONUNBUFFERED 1

# Directorio de trabajo dentro del contenedor
WORKDIR /app

# Instalamos dependencias del sistema necesarias para PostgreSQL
RUN apt-get update \
    && apt-get install -y --no-install-recommends gcc libpq-dev \
    && apt-get clean

# Copiamos las dependencias e instalamos
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# Copiamos el resto del código
COPY ..

# Comando para iniciar FastAPI
CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]
```

## B. El Frontend ([/frontend/Dockerfile](#))

Asumiendo que usas Next.js o React. Este es un setup robusto para desarrollo/producción.

```
Dockerfile
# Usamos Node.js versión Alpine (muy ligera)
FROM node:18-alpine

WORKDIR /app

# Copiamos package.json primero para aprovechar la caché de Docker
COPY package*.json .

# Instalamos dependencias
RUN npm install

# Copiamos el código fuente
COPY ..

# Construimos la aplicación (Build de producción)
RUN npm run build
```

```
# Exponemos el puerto (3000 es estándar en React/Next)
EXPOSE 3000
```

```
# Iniciamos la app
CMD ["npm", "start"]
```

---

## Paso 3: El Orquestador (`docker-compose.yml`)

Crea este archivo en la raíz del proyecto. Aquí es donde definimos cómo se hablan los sistemas y configuramos la base de datos real (PostgreSQL).

YAML

```
version: '3.8'
```

```
services:
```

```
# 1. BASE DE DATOS (PostgreSQL)
db:
  image: postgres:15-alpine
  container_name: nexo_db
  volumes:
    - postgres_data:/var/lib/postgresql/data # Persistencia de datos
  environment:
    - POSTGRES_USER=nexo_admin
    - POSTGRES_PASSWORD=nexo_secure_pass
    - POSTGRES_DB=nexo_sinergico_db
  ports:
    - "5432:5432" # Accesible localmente si necesitas ver la DB
  networks:
    - nexo_network
```

```
# 2. BACKEND (FastAPI - Python)
```

```
backend:
  build: ./backend
  container_name: nexo_backend
  command: uvicorn main:app --host 0.0.0.0 --port 8000 --reload
  volumes:
    - ./backend:/app # "Hot Reload": Los cambios en tu código se reflejan al instante
  ports:
    - "8000:8000"
  environment:
    # La URL mágica: 'db' es el nombre del servicio de arriba
```

```

- DATABASE_URL=postgresql://nexo_admin:nexo_secure_pass@db/nexo_sinergico_db
- SECRET_KEY=tu_clave_super_secreta_jwt
depends_on:
- db
networks:
- nexo_network

# 3. FRONTEND (React/Next.js)
frontend:
build: ./frontend
container_name: nexo_frontend
ports:
- "3000:3000"
environment:
# El navegador del usuario necesita saber dónde está la API
- NEXT_PUBLIC_API_URL=http://localhost:8000
volumes:
- ./frontend:/app
- /app/node_modules
depends_on:
- backend
networks:
- nexo_network

volumes:
postgres_data:

networks:
nexo_network:
driver: bridge

```

---

## Paso 4: Ajuste Crítico en el Código Python

Hasta ahora usabas SQLite (`sqlite:///...`). Ahora usas PostgreSQL profesional. Debes hacer un pequeño cambio en tu archivo `main.py` (o `database.py`) para que lea la variable de entorno que definimos en Docker.

**En `backend/main.py`:**

```

Python
import os
# ... tus imports ...

```

```
# ANTES:  
# DATABASE_URL = "sqlite:///nexo_sinergico.db"  
  
# AHORA (Busca la variable de entorno, si no está, usa SQLite por defecto):  
DATABASE_URL = os.getenv(  
    "DATABASE_URL",  
    "sqlite:///nexo_sinergico.db"  
)  
  
# Nota: Si usas PostgreSQL, necesitas instalar el driver:  
# pip install psycopg2-binary
```

(Recuerda añadir `psycopg2-binary` a tu `requirements.txt`)

---

## Paso 5: ¡Despegue!

Abre tu terminal en la carpeta raíz y ejecuta:

Bash  
docker-compose up --build

### Lo que verás:

1. Docker descargará Postgres, Python y Node.
2. Instalará todas las librerías automáticamente.
3. Verás los logs de los 3 colores (DB, Backend, Frontend) mezclados en la consola.

### Resultado:

- Frontend: <http://localhost:3000>
- Backend Docs: <http://localhost:8000/docs>
- Base de Datos: Corriendo en puerto 5432.

## Ventaja Oculta de esta Opción

Al usar `volumes` en el `docker-compose` (líneas de "Hot Reload"), puedes seguir editando tu código en VS Code y los cambios se verán reflejados en el contenedor **sin tener que reiniciar**. Tienes la robustez de producción con la velocidad de desarrollo.