Esta es la decisión que separa a los aficionados de los profesionales!

Manejar las credenciales en un archivo `.env` (Variables de Entorno) es la **Best Practice #1** en seguridad DevOps. Evita que tus contraseñas terminen en GitHub por accidente y facilita cambiar configuraciones entre "Desarrollo" y "Producción" sin tocar el código.

Aquí tienes la implementación paso a paso para el **Nexo Sinérgico**.

Paso 1: El Archivo Secreto (`.env`)

Crea un archivo llamado `.env` en la **raíz** de tu proyecto (al mismo nivel que `docker-compose.yml`).

Contenido de `.env`:

Properties
```
# --- CONFIGURACIÓN DE BASE DE DATOS ---
POSTGRES_USER=nexo_admin
POSTGRES_PASSWORD=tu_password_super_seguro_y_largo_123
POSTGRES_DB=nexo_sinergico_db
POSTGRES_HOST=db
POSTGRES_PORT=5432

# --- SEGURIDAD DE LA APLICACIÓN (BACKEND) ---
# Genera esto con: openssl rand -hex 32
SECRET_KEY=e8f9a0...pegue_aqui_una_cadena_larga_y_aleatoria...
ALGORITHM=HS256
ACCESS_TOKEN_EXPIRE_MINUTES=60

# --- FRONTEND (URL PÚBLICA) ---
NEXT_PUBLIC_API_URL=http://localhost:8000
```

Paso 2: El Guardián del Git (`.gitignore`)

Esto es **CRÍTICO**. Debes decirle a Git que ignore este archivo para que nunca se suba al repositorio.

Abre (o crea) el archivo `.gitignore` en la raíz y añade:

Plaintext
```
# Ignorar variables de entorno y secretos
```

```
.env
.env.local
.venv/
__pycache__/

# Mantener un ejemplo seguro
!.env.example
```

Tip Pro: Crea un archivo llamado `.env.example` con las mismas claves pero valores vacíos o falsos (`POSTGRES_PASSWORD=cambiar_esto`). Ese sí se sube a Git para que otros desarrolladores sepan qué variables configurar.

Paso 3: Actualizar `docker-compose.yml`

Ahora modificamos el orquestador para que lea automáticamente el archivo `.env` y pase los valores a los contenedores.

YAML

```
version: '3.8'

services:
  # 1. BASE DE DATOS
  db:
    image: postgres:15-alpine
    container_name: nexo_db
    volumes:
      - postgres_data:/var/lib/postgresql/data
    environment:
      # Docker lee las variables del archivo .env automáticamente
      - POSTGRES_USER=${POSTGRES_USER}
      - POSTGRES_PASSWORD=${POSTGRES_PASSWORD}
      - POSTGRES_DB=${POSTGRES_DB}
    ports:
      - "${POSTGRES_PORT:-5432}:5432" # Usa 5432 por defecto si no está en .env
    networks:
      - nexo_network

  # 2. BACKEND
  backend:
    build: ./backend
    container_name: nexo_backend
```

```
    command: uvicorn main:app --host 0.0.0.0 --port 8000 --reload
    volumes:
      - ./backend:/app
    ports:
      - "8000:8000"
    environment:
      # Pasamos las variables explícitamente o usamos env_file
      -
DATABASE_URL=postgresql://${POSTGRES_USER}:${POSTGRES_PASSWORD}@${POST
GRES_HOST}:${POSTGRES_PORT}/${POSTGRES_DB}
      - SECRET_KEY=${SECRET_KEY}
      - ALGORITHM=${ALGORITHM}
      - ACCESS_TOKEN_EXPIRE_MINUTES=${ACCESS_TOKEN_EXPIRE_MINUTES}
    depends_on:
      - db
    networks:
      - nexo_network

  # ... (Frontend igual) ...
```

Paso 4: Centralizar Configuración en Python (Backend)

En lugar de usar `os.getenv` disperso por todos lados, la **Best Practice** es tener un solo archivo de configuración.

1. Crea un archivo nuevo: `backend/config.py`.
2. Usa `pydantic-settings` (estándar moderno) o `os` simple. Usaremos `os` para no añadir dependencias complejas ahora, pero centralizado.

Archivo `backend/config.py`:

Python
```
import os

class Settings:
    PROJECT_NAME: str = "Nexo Sinérgico"
    PROJECT_VERSION: str = "2.5.0"

    # Base de Datos
    # Si no hay variable (ej. local sin docker), usa SQLite como fallback
    DATABASE_URL: str = os.getenv(
        "DATABASE_URL",
```

```
        "sqlite:///./nexo_local.db"
    )

    # Seguridad JWT
    SECRET_KEY: str = os.getenv("SECRET_KEY",
"super_secreto_fallback_inseguro_para_dev")
    ALGORITHM: str = os.getenv("ALGORITHM", "HS256")
    ACCESS_TOKEN_EXPIRE_MINUTES: int =
int(os.getenv("ACCESS_TOKEN_EXPIRE_MINUTES", 60))

# Instancia global
settings = Settings()
```

Actualiza `backend/main.py`:

Ahora tu código principal queda mucho más limpio:

```
Python
# Imports
from config import settings # <--- Importamos la configuración centralizada

# ...

# Uso en la DB
engine = create_engine(settings.DATABASE_URL)

# Uso en JWT
def create_access_token(data: dict):
    # ...
    encoded_jwt = jwt.encode(to_encode, settings.SECRET_KEY,
algorithm=settings.ALGORITHM)
    return encoded_jwt
```

Resumen de la Mejora

Con estos cambios, acabas de blindar el **Nexo Sinérgico**:

1. **Portabilidad:** Si mueves el proyecto a un servidor de producción, solo creas un `.env` nuevo con la contraseña real. No tocas ni una línea de código.
2. **Seguridad:** Tus contraseñas ya no viven en el código fuente.
3. **Orden:** `config.py` actúa como la "Torre de Control" de todas tus variables.

¿Todo listo? Ejecuta de nuevo `docker-compose up --build` y verifica que el sistema levante leyendo las nuevas credenciales. ¡Ya tienes infraestructura de nivel empresarial!