

Manual: Dockerizar Proyecto Django en Windows 11

Prerrequisitos del Sistema

1. Instalar WSL (Windows Subsystem for Linux)

Desde PowerShell como Administrador:

```
powershell  
  
wsl --install
```

Después de la instalación, reiniciar Windows. El sistema te guiará para configurar Ubuntu.

2. Verificar Instalación de WSL

```
powershell  
  
# Verificar versión  
wsl --version  
  
# Ver distribuciones instaladas  
wsl --list --verbose
```

3. Instalar Docker Desktop

1. Descargar Docker Desktop desde: <https://docs.docker.com/desktop/install/windows-install/>
2. Durante la instalación, marcar "Use WSL 2 instead of Hyper-V"
3. Reiniciar si es necesario

4. Configurar Integración WSL-Docker

1. Abrir Docker Desktop
2. Ir a Settings → Resources → WSL Integration
3. Activar "Enable integration with my default WSL distro"
4. Activar "Ubuntu"
5. Apply & Restart

5. Verificar Configuración

Desde Ubuntu (WSL):

```
bash
```

```
docker --version
docker ps
docker run hello-world
```

Preparación del Proyecto Django

1. Acceder al Proyecto desde WSL

```
bash

# Crear carpeta proyectos
mkdir ~/proyectos
cd ~/proyectos

# Clonar o copiar proyecto Django
# El proyecto debe tener: manage.py, requirements.txt, settings.py
```

2. Verificar Estructura del Proyecto

```
bash

ls -la
# Debe contener:
# - manage.py
# - requirements.txt
# - carpeta de configuración Django
# - apps del proyecto
```

Configuración Docker

1. Crear Dockerfile

En la raíz del proyecto:

```
dockerfile
```

FROM python:3.11

Variables de entorno

ENV PYTHONDONTWRITEBYTECODE 1

ENV PYTHONUNBUFFERED 1

Directorio de trabajo

WORKDIR /app

Instalar dependencias del sistema para PostgreSQL

RUN apt-get update \

... && apt-get install -y --no-install-recommends \

..... postgresql-client \

..... gcc \

..... python3-dev \

..... libpq-dev \

..... build-essential \

... && rm -rf /var/lib/apt/lists/*

Instalar dependencias Python

COPY requirements.txt .

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

Copiar código

COPY ..

Puerto de exposición

EXPOSE 8000

Comando por defecto

CMD ["python", "manage.py", "runserver", "0.0.0.0:8000"]

2. Crear docker-compose.yml

yml

version: '3.8'

services:

Django Application

django:

build: .

container_name: proyecto_django

ports:

- "8000:8000"

volumes:

- ./app

environment:

- DEBUG=1

env_file:

- .env

depends_on:

- postgres

networks:

- proyecto_network

PostgreSQL Database

postgres:

image: postgres:15

container_name: proyecto_postgres

environment:

POSTGRES_DB: proyecto_db

POSTGRES_USER: django_user

POSTGRES_PASSWORD: django_password

ports:

- "5432:5432"

volumes:

- postgres_data:/var/lib/postgresql/data

networks:

- proyecto_network

pgAdmin para gestionar PostgreSQL

pgadmin:

image: dpage/pgadmin4

container_name: proyecto_pgadmin

environment:

PGADMIN_DEFAULT_EMAIL: admin@admin.com

PGADMIN_DEFAULT_PASSWORD: admin

ports:

- "8080:80"

depends_on:

- postgres

```
... networks:
...   - proyecto_network
```

volumes:

```
... postgres_data:
```

networks:

```
... proyecto_network:
...   driver: bridge
```

3. Crear archivo .env

```
bash

# Django settings
DEBUG=True
SECRET_KEY=tu-secret-key-para-desarrollo

# Database settings
DB_ENGINE=django.db.backends.postgresql
DB_NAME=proyecto_db
DB_USER=django_user
DB_PASSWORD=django_password
DB_HOST=postgres
DB_PORT=5432

# Allowed hosts
ALLOWED_HOSTS=localhost,127.0.0.1
```

4. Actualizar settings.py de Django

Modificar el archivo de configuración para usar variables de entorno:

```
python
```

```
import os
from dotenv import load_dotenv

# Cargar variables del archivo .env
load_dotenv()

# Configuración de base de datos
DATABASES = {
    .... 'default': {
        .... 'ENGINE': 'django.db.backends.postgresql',
        .... 'NAME': os.getenv('DB_NAME'),
        .... 'USER': os.getenv('DB_USER'),
        .... 'PASSWORD': os.getenv('DB_PASSWORD'),
        .... 'HOST': os.getenv('DB_HOST'),
        .... 'PORT': os.getenv('DB_PORT'),
        .... }
    }

# Hosts permitidos
ALLOWED_HOSTS = ['localhost', '127.0.0.1', '0.0.0.0']
```

5. Crear .dockerignore

```
bash

__pycache__
*.pyc
.git
.gitignore
README.md
.env.example
node_modules
.vscode
.idea
*.log
```

Ejecutar el Proyecto

1. Construir y Levantar Servicios

```
bash

# Desde el directorio del proyecto
docker-compose up --build -d
```

2. Ejecutar Migraciones

```
bash

# Aplicar migraciones de Django
docker-compose exec django python manage.py migrate

# Verificar migraciones aplicadas
docker-compose exec django python manage.py showmigrations
```

3. Crear Superusuario

```
bash

docker-compose exec django python manage.py createsuperuser
```

Verificación y Uso

URLs Disponibles

- **Aplicación Django:** <http://localhost:8000>
- **Admin Django:** <http://localhost:8000/admin>
- **pgAdmin:** <http://localhost:8080> (admin@admin.com / admin)

Comandos Útiles

```
bash

# Ver contenedores corriendo
docker-compose ps

# Ver logs
docker-compose logs django

# Parar servicios
docker-compose down

# Reiniciar servicios
docker-compose restart

# Ejecutar comandos Django
docker-compose exec django python manage.py [comando]

# Acceder al contenedor
docker-compose exec django bash
```

Configurar pgAdmin

1. Acceder a <http://localhost:8080>
2. Login con admin@admin.com / admin
3. Agregar servidor:
 - Name: Proyecto
 - Host: postgres
 - Port: 5432
 - Database: proyecto_db
 - Username: django_user
 - Password: django_password

Flujo de Trabajo Diario

```
bash

# Iniciar proyecto
cd ~/proyectos/mi-proyecto
docker-compose up -d

# Trabajo de desarrollo...

# Parar proyecto
docker-compose down
```

Solución de Problemas Comunes

Error "no configuration file provided"

- Verificar que estás en el directorio correcto del proyecto
- El archivo docker-compose.yml debe estar presente

Error "relation does not exist"

- Ejecutar migraciones: `docker-compose exec django python manage.py migrate`

Puerto ocupado

- Cambiar puertos en docker-compose.yml
- Verificar puertos en uso: `netstat -tulpn | grep :8000`

Error de conexión a base de datos

- Verificar variables en archivo .env

- Revisar logs: `docker-compose logs postgres`

Ventajas de la Dockerización

- **Aislamiento:** Cada proyecto en su propio entorno
- **Reproducibilidad:** Mismo entorno en todas las máquinas
- **Facilidad de setup:** Un comando levanta todo el stack
- **Limpieza:** No ensuciar la máquina host
- **Escalabilidad:** Fácil agregar más servicios

Este manual permite dockerizar cualquier proyecto Django existente y mantener un flujo de desarrollo consistente.