Manual de Proyecto DRF con Front End React Apache Docker

Instalación del Proyecto

Pre-Requisitos

Conexión a Internet

Proyecto Django Rest Framework

Front End React

Motor de Contenedores

Este proyecto requiere de un motor de contenedores, **se recomienda Docker y se asumirá que este es el motor en uso por el resto de este documento**, la instalación de este depende del sistema operativo en uso. Se recomienda seguir las instrucciones de la documentación oficial de **Docker**. También funciona con **Podman**.

https://docs.docker.com/engine/install/

Nota: **La documentación de Docker solo está en Inglés**, use un traductor de ser necesario.

Python de Django

Proyecto de **Django** con versión de **Python** preferentemente **3.12.x**, las versiones **3.11.x** y **3.9.x** requieren de configuración, cualquier otra versión no está soportada para este proyecto.

También se espera que tu proyecto tenga un **ambiente de Python venv** incluido.

Preparaciones

Construir Imagen para Contenedor

Este proyecto usa una imagen personalizada basada en **AlmaLinux 9.4** y corre **Python 3.12**, para que el proyecto de Django corra correctamente, **las versiones mayores de Python (X.XX.x) dentro del contenedor y la usada por el proyecto Django tienen que ser las mismas.** Las versiones de **Python** oficialmente disponibles en esta versión de **AlmaLinux** son **3.12, 3.11 y 3.9**, cualquier otra no está cubierta por este manual, para saber que versión de Python usas, usa "`python --version`" dentro del ambiente de Python de tu proyecto Django.

A este punto se asume que tienes **Docker** instalado y está en tu PATH (El instalador de Docker debería haber hecho esto automaticamente), abre un terminal y **usa este comando:**

Asegurate de cambiar el argumento "PY_VER" y la versión de Python indicada en el tag de la imagen ("python-3-12") por la versión que usa tu proyecto de Django, recuerda el tag de la imagen (Todo lo que viene despues de "-t"), puedes usar este documento para tomar nota.

```
docker build --build-arg PY_VER="3.12" -t usuario-local/apache-wsgi-almalinux:9.4-python3-12 .
```

Esto construirá la imagen que usaremos para generar nuestro contenedor, tomará un rato ya que descargará primero la imagen de AlmaLinux 9.4 y luego descargará e instalará dentro la imagen las herramientas que necesitaremos para correr Apache dentro y Django dentro del contendedor que generemos.

Configurar Django

El proyecto Django debe estar configurado de cierta manera.

Edita el archivo **"settings.py"** de tu proyecto y añade lo siguiente al final del archivo:

```
STATIC_ROOT = BASE_DIR / 'static'
```

Esto especifica la carpeta de contenido **estático** que Apache usará para servir el contenido Web de Django de manera correcta, pero aún debemos generar ese contenido.

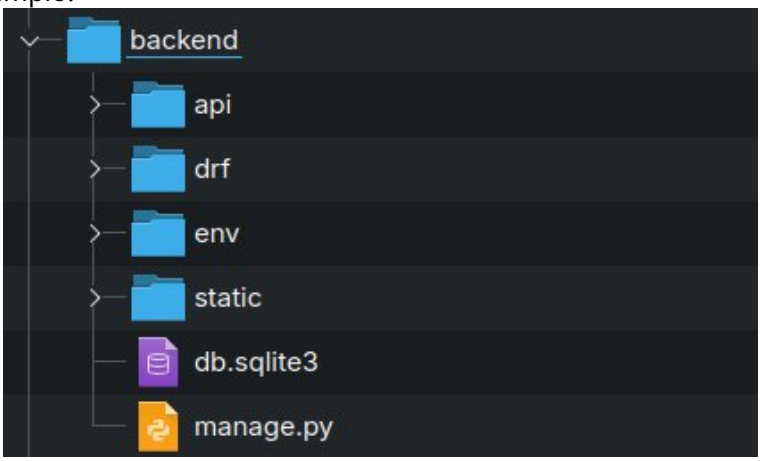
Entra al **ambiente virtual** de Python y **usa este comando** para generar el contenido **estático**.

```
python ./manage.py collectstatic
```

Una vez finalizado el proceso de colección, deberian haber una nueva carpeta en tu proyecto de Django llamada "static".

Ubica tu Proyecto de Django en el Proyecto

Ubica el **contenido** de tu proyecto de Django en la carpeta "**backend**", abajo aparece un proyecto de ejemplo.



Configurar VirtualHost de Apache

Usaremos un **VirtualHost** de Apache para servir el contenido de Django, ya hay una plantilla en la carpeta **"apache-confs".**
En la carpeta **"apache-confs"**, edita el archivo plantilla y reemplaza todas las instancias de "**CARPETA-DJANGO-REST-FRAMEWORK**" con el nombre de la **carpeta de proyecto DRF** de tu proyecto Django, esta es la carpeta que contiene el archivo **"wsgi.py"** y **"settings.py"**, detalles sobre que hace cada parametro estan en los comentarios.

Prueba e Iniciación del Servidor

Ahora que Django debiera estar listo detrás de Apache, **queda probar que el servidor funciona** correctamente, es mejor probar que este sea el caso antes de preparar nuestro Front End.

Para esto, usaremos Docker Compose, ya hay una plantilla en el proyecto llamada "**compose.yml**".

Asegurate que este archivo tenga sus parámetros deseados, por ejemplo, si cambiaste el nombre de la imagen o su tag, tendrás que apuntar el archivo **compose** ahí mediante la variable "**image:**".

Inicia un terminal en la carpeta raíz del proyecto y usa este comando (En algunos sistemas puede que tengas que usar "docker**-**compose").

```
docker compose up -d
```

Usando **Docker**, usamos la función "**compose",** y levantameros (**up**) un contenedor, esto basará el contenedor en los parametros que tenemos en nuestro archivo "**compose.yml",** por defecto, la función **compose** buscará un archivo de nombre "**compose**" o "**docker-compose**" de formato "**.yml**" o "**.yaml**". El argumento "**-d**" es para "desacoplarnos" (**detach**) del proceso de creación del contenedor, de otra forma, nuestro terminal se usaría para correr el proceso de docker compose.
Por defecto servimos Django en el puerto 8000, intenta acceder desde el navegador con http://localhost:8000/[ruta_configurada].

Solución de Errores y Verificación de Salud

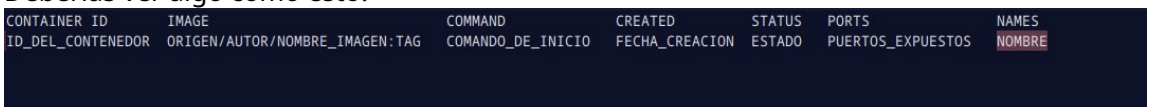
Si aparece algún error, entonces algo malo sucedió al crear el contenedor y tendrá que ver con la configuración de nuestro archivo "**compose.yml**", si es así, verifica todo, incluyendo si la imagen del contenedor está presente.
Si solo aparece el nombre del contenedor creador y algunos IDs, significa que el contenedor se creó correctamente, pero no significa que este esté funcionando, verifícalo con el siguiente comando.

```
docker ps
```

Esto mostrará los contenedores actualmente corriendo, si no aparece nada, tu contenedor se hace apagado al iniciar, la imagen que creamos está configurada para apagarse si Apache se cierra, en dicho caso, puedes leer el informe de Apache antes de apagarse con el comando de Docker "**logs**", pero para usarlo necesitamos el nombre de nuestro contenedor, el siguiente comando sirve para listas todos los contenedores que tengamos, estén corriendo o no:

```
docker ps --all
```

Deberias ver algo como esto:



Una vez sepas el nombre de tu contenedor, podrás ver su informe con este comando:

```
docker logs [NOMBRE_DEL_CONTENEDOR]
```

Si Apache no está funcionando como debería, esto tendría que darte una pista a que es lo que está pasando, a este punto lo más probable es que sea un problema con tu **VirtualHost** o Backend.

Una vez que hayas aplicado una posible solución, intenta reiniciar el contenedor:

```
docker start [NOMBRE_DEL_CONTENEDOR] #Inicia contenedor

docker stop [NOMBRE_DEL_CONTENEDOR] #Detiene contenedor

docker restart [NOMBRE_DEL_CONTENEDOR] #reinicia contenedor
```

Si modificaste el archivo "**compose.yml**" tendrás que usar `docker compose up -d` de nuevo, esto regenerará e iniciará el contenedor, luego, verifica que tu contenedor esté corriendo con `docker ps`, si eso falla, intenta buscar otra solución e inténtalo de nuevo.

Para errores de Apache, ve a la carpeta "**logs**" del proyecto, esta es una montura de la carpeta "**logs"** de **Apache**, usa los informes para encontrar problemas con tu servidor de Apache.

Preparar Front End React

Ahora toca preparar nuestro Front End React, esto debería resultar más sencillo.

Construir Versión de Proyecto React para Producción

Antes de servir nuestro Front End hay que construir una versión apta para producción, esto es simple.

Entra a la raiz de tu proyecto React y ejecuta el siguiente comando:

```
npm run build
```

Esto contruirá una versión de producción de tu proyecto en la carpeta "**build**", esto puede tomar un tiempo, una vez listo, tendrás una versión de tu proyecto lista para usar en servidores web.

Servir Front End desde Apache

Copia los contenidos de la carpeta "**build**" de tu proyecto de React a la carpeta "**frontend**" del proyecto Docker, esto montará tu Front End en "**/var/www/html**" dentro del contenedor, Apache sirve por defecto el contenido de este directorio en el puerto 80, recuerda que este puerto lo exponemos en el puerto 3000.

Luego, reinicia el contenedor y verifica que el Front End funciona desde tu navegador, http://localhost:3000/

Solucionar Problema CORS

Lo más probable es que tu Front End no pueda conectarse a tu Back End, para solucionar esto tienes que cambiar la politica CORS de tu proyecto de Django, esta politica define los dominios que pueden hacer solicitudes a nuestro servicio, ve a tu proyecto de Django y entra a tu ambiente virtual de Python con un terminal, luego, instala "**django-cors-headers**" con **pip.**

```
pip install django-cors-headers
```

Ahora que tenemos el paquete necesario, declaremos la politica en nuestro proyecto, ve a "**setttings.py**" y haz las siguientes modificaciones:

Añade "**corsheaders**" a tus apps instaladas:

```
INSTALLED_APPS = [
    ...,
    "corsheaders",
    ...,
]
```

Añade "**corsheaders.middleware.CorsMiddleware**" a tu Middleware

```
MIDDLEWARE = [
    ...,
    "corsheaders.middleware.CorsMiddleware",
    "django.middleware.common.CommonMiddleware",
    ...,
]
```

Declara la politica de **CORS** para permitir acceso a cualquier:

```
CORS_ORIGIN_ALLOW_ALL = True
```

Una vez hecho esto, tu Front End debería ser capaz de usar la API de tu proyecto Django.
Para más información sobre CORS en Django vea la documentación del paquete **django-cors-headers** en https://pypi.org/project/django-cors-headers/ .