Curso de Django

Módulo 5: Docker.

Lcdo. Diego Medardo Saavedra García. Mgtr.

2023-08-03

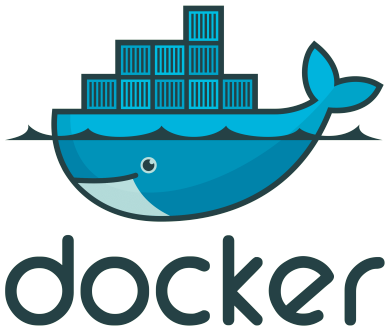
Tabla de contenidos

1 Módulo 5:Docker.

1.1 Fundamentos de Docker.

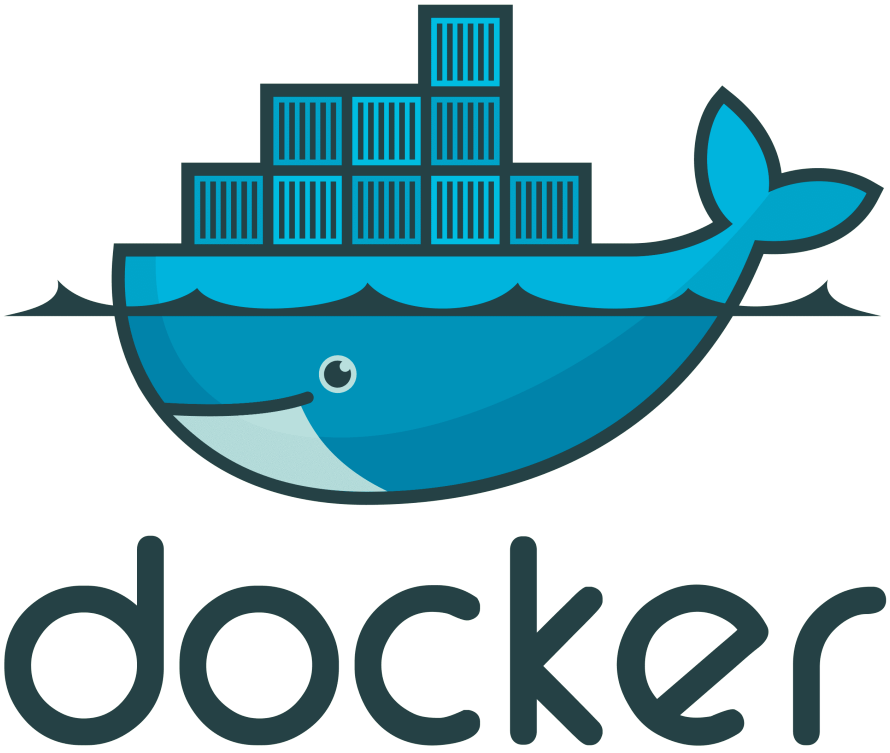
1.1.1 ¿Qué es Docker?

Docker es una tecnología que se ha convertido en el estandar para desarrollar, distribuir y ejecutar aplicaciones a nivel productivo en todo el mundo, en la nube, un nuestras máquinas,



es decir en todos lados.

1.1.2 ¿Por qué utilizar Docker?



Imaginemos un proyecto en el que muchas personas (decenas de miles) puedan colaborar en una especie de pizarra digital para compartir nuestras ideas y colaborar, para poder escalar a cientos de miles o millones de usuarios existen muchos desafíos, esto se refiere a lo que sucede

con el software cuando lo escribimos en nuestras máquinas y lo que debe hacer cuando este en los dispositivos, los servidores, la nube, etc.

Estos son los problemas a los que nos enfrenteamos los desarrolladores en todo el mundo, sin importar el lenguaje de programación, framework, base de datos o servidor que utilicemos.

Docker nos permite resolver este tipo de problemas triviales de forma sencilla, rápida y eficiente. Ayuda a trabajar de forma rápida, mejor con mayor confianza y seguridad.

1.2 Problemas que están presentes en el Desarrollo de Software.

1. **Construir:** Necesitamos saber como se construye el software que estamos desarrollando.
2. **Distribuir:** Como se va a distribuir a los usuarios que van a utilizar el software
3. **Ejecutar:** Como va a funcionar el sofware que hemos desarrollado.

1. Construir.- Escribir código en la máquina del desarrollador. (Compile, que no compile, arreglar el bug, compartir código, etc.)

Problemática:

- Entorno de desarrollo (paquetes)
- Dependencias (Frameworks, bibliotecas)
- Versiones de entornos de ejecución (runtime, versión Node)
- Equivalencia de entornos de desarrollo (compartir el código)
- Equivalencia con entornos productivos (pasar a producción)
- Servicios externos (integración con otros servicios ej: Base de Datos)

2. Distribuir.- Llevar la aplicación donde se va a desplegar (Transformarse en un artefacto).

Problemática:

- Output de build heterogeneo (múltiples compilaciones)
- Acceso a servidores productivos (No tenemos acceso al servidor)
- Ejecución nativa vs virtualizada
- Entornos Serverless

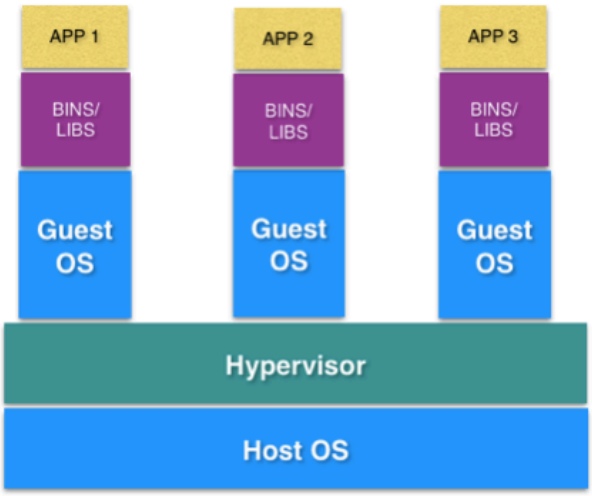
3. Ejecutar.- Implementar la solución en el ambiente de producción (Subir a producción).

Hacer que funcione como debería funcionar.

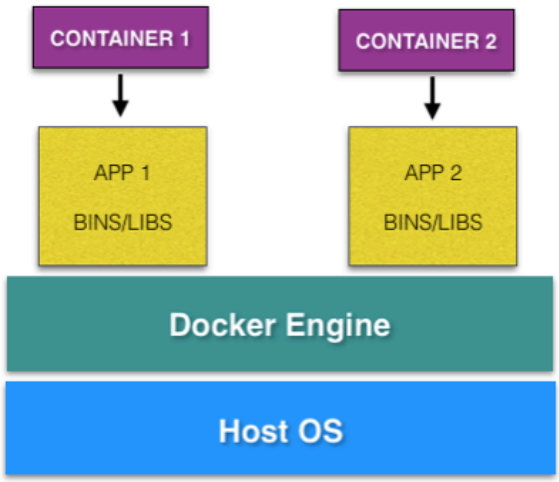
Problemática:

- Dependencia de aplicación (paquetes, runtime)
- Compatibilidad con el entorno productivo (sistema operativo poco amigable con la solución)
- Disponibilidad de servicios externos (Acceso a los servicios externos)
- Recursos de hardware (Capacidad de ejecución - Menos memoria, procesador más debil).

1.3 Maquinas Virtuales vs Docker.



VIRTUAL MACHINE ARCHITECTURE



DOCKER ARCHITECTURE

1.3.1 Virtualización

Version virtual de algún recurso tecnológico, como un servidor, un dispositivo de almacenamiento, un sistema operativo o recurso de red.

La virtualización resuelve los tres problemas de la sección anterior.

1.3.2 Máquinas Virtuales.

1.3.3 Problemas de las VMs

Peso:

- En el orden de los Gb. Repiten archivos en común.
- Inicio Lento

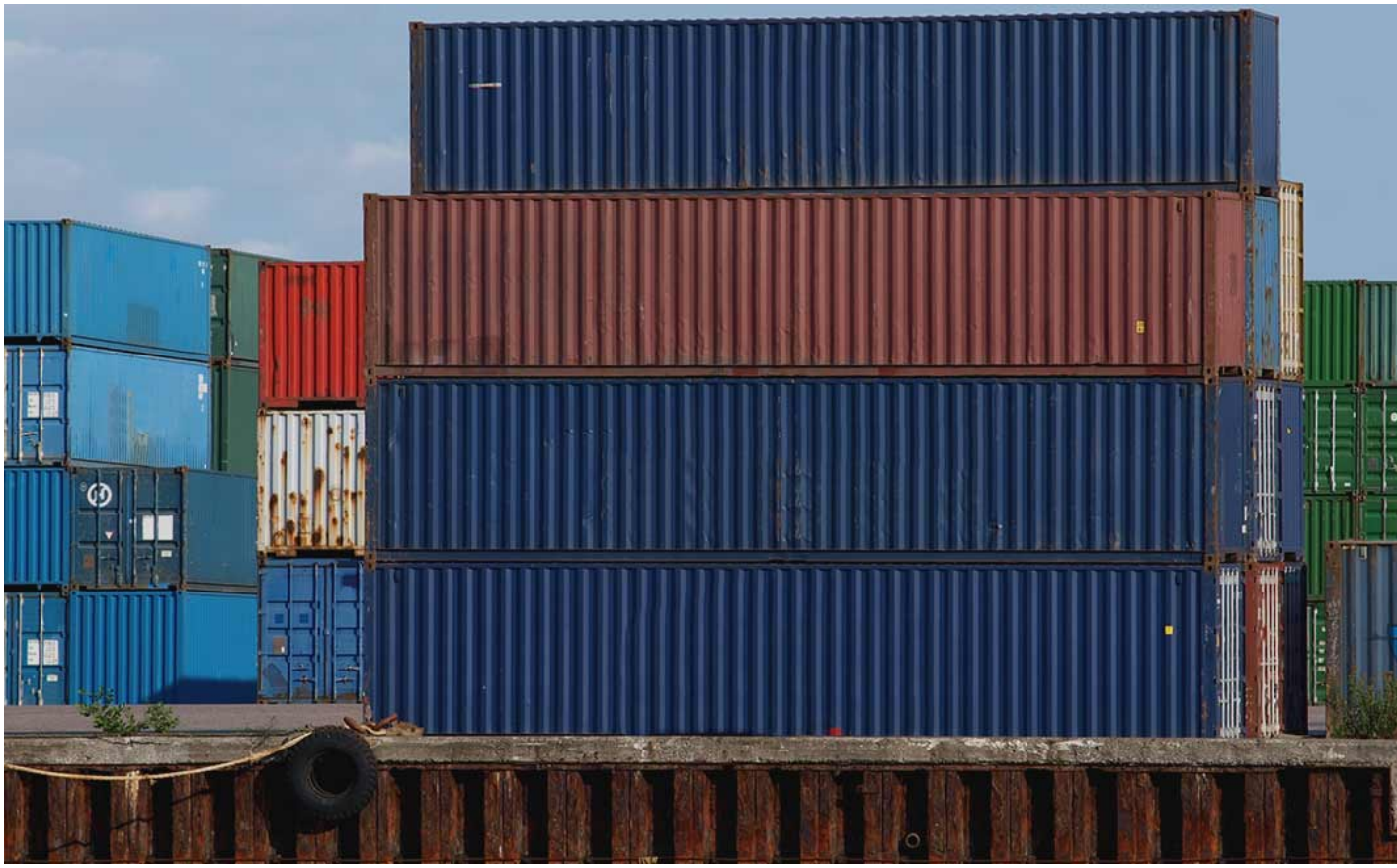
Costo de administración:

- Necesita mantenimiento igual que cualquier otra comptuadora.

Múltiples formatos.

- VDI, VMDK, VHD, raw, entre otros.

1.3.4 Contenedores.



¿Qué es un contenedor?

- Es una agrupación de procesos.
- Es una entidad lógica, no tiene el limite estricto de las máquinas virtuales, emulación del sistema operativo simulado por otra más abajo.
- Ejecuta sus procesos de forma nativa.
- Los procesos que se ejecutan adentro de los contenedores ven su universo como el contenedor lo define, no pueden ver mas allá del contenedor, a pesar de estar corriendo en una maquina más grande.
- No tienen forma de consumir más recursos que los que se les permite. Si esta restringido en memoria ram por ejemplo, es la única que pueden usar.
- A fines prácticos los podemos imaginar cómo maquinas virtuales, pero NO lo son. Máquinas virtuales livianas.

- Docker corre de forma nativa solo en Linux.
- Sector del disco: Cuando un contenedor es ejecutado, el daemon de docker le dice, a partir de acá para arriba este disco es tuyo, pero no puedes subir mas arriba.
- Docker hace que los procesos adentro de un contenedor este aislados del resto del sistema, no le permite ver más allá.
- Cada contenedor tiene un ID único, también tiene un nombre.

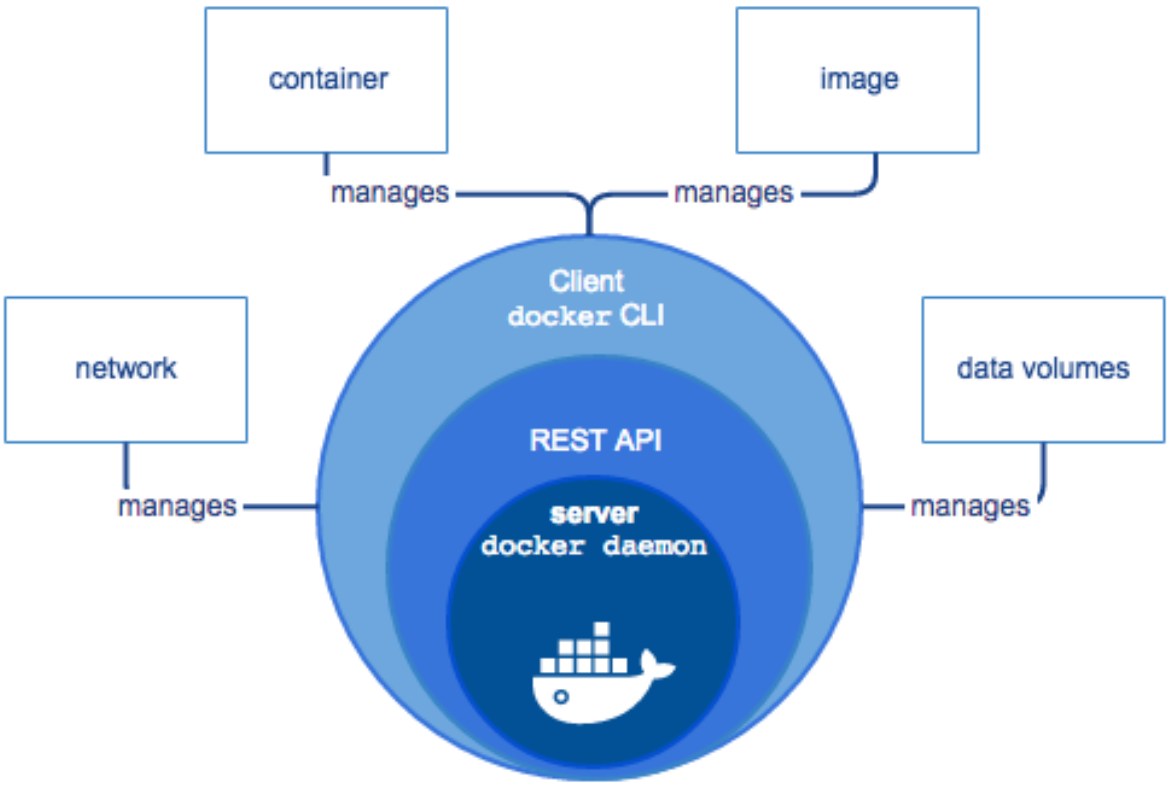
1.3.5 Containerización.

Empleo de **contenedores** para construir y desplegar software.

Caracteristicas.

- Flexibles
- Livianos
- Portables
- Bajo acoplamiento
- Escalables
- Seguros

1.4 Arquitectura de Docker.



Docker es una Plataforma que permite construir, ejecutar y compartir aplicaciones mediante **contenedores**.

Container: Es una instancia de una imagen.

Image: Es un paquete de software que incluye todo lo necesario para ejecutar una aplicación.

Network: Es una red virtual que permite la comunicación entre contenedores y con el exterior.

Data Volumes: Es un directorio que se monta en un contenedor para que este pueda acceder a el.

1.5 Trabajar con Docker.

Paso 1: Instalar Docker

Lo primero que debemos hacer es instalar Docker en nuestro sistema operativo. Para ello, visita el sitio web oficial de Docker y sigue las instrucciones de instalación para tu sistema operativo específico.

Paso 2: Verificar la Instalación

Una vez instalado Docker, verifica que esté funcionando correctamente ejecutando el siguiente comando:

```
docker --version
```

Este comando mostrará la versión de Docker instalada en tu sistema.

Ahora empecemos a probar Docker.

```
docker run hello-world
```

Este comando descargará la imagen de Docker **hello-world** y la ejecutará en un contenedor. Cuando el contenedor se ejecute, imprimirá un mensaje y saldrá.

Paso 3: Descargar una Imagen de Docker

Docker utiliza imágenes para crear contenedores. Una imagen es una plantilla de solo lectura que contiene el sistema operativo, el software y los archivos necesarios para ejecutar una aplicación. Puedes descargar imágenes de Docker Hub, que es un registro de imágenes públicas disponibles para su uso.

Para descargar una imagen, utiliza el siguiente comando:

```
docker pull nombre_de_la_imagen
```

Por ejemplo, para descargar la imagen oficial de Ubuntu, puedes ejecutar:

```
docker pull ubuntu
```

Paso 4: Ver Imágenes Descargadas

Para ver la lista de imágenes descargadas en tu sistema, puedes utilizar el siguiente comando:

```
docker images
```

Paso 5: Crear un Contenedor

Una vez que tengas una imagen descargada, puedes crear un contenedor basado en esa imagen. Para ello, utiliza el siguiente comando:

```
docker run nombre_de_la_imagen
```

Por ejemplo, para crear un contenedor basado en la imagen de Ubuntu, puedes ejecutar:

```
docker run ubuntu
```

Paso 6: Ver Contenedores en Ejecución

Para ver la lista de contenedores en ejecución en tu sistema, puedes utilizar el siguiente comando:

```
docker ps
```

Paso 7: Ver Todos los Contenedores

Para ver la lista de todos los contenedores, incluidos los que no están en ejecución, puedes utilizar el siguiente comando:

```
docker ps -a
```

Ahora probemos el docker id con el comando inxpect.

```
docker inspect ID_del_contenedor
```

Con este comando vemos toda la información del contenedor.

Ahora probemos dar un NAME al contenedor.

```
docker run --name nombre_del_contenedor nombre_de_la_imagen
```

Muy bien, por favor hagamos una pequeña práctica para probar lo que hemos aprendido, por favor creemos 10 contenedores con nombres de paises, por ejemplo: Colombia, Argentina, Brasil, Chile, Ecuador, España, Francia, Italia, México, Perú.

```
docker run --name Colombia hello-world
docker run --name Argentina hello-world
docker run --name Brasil hello-world
docker run --name Chile hello-world
'''etc'''
```

Ahora probemos el comando inxpect con el nombre del contenedor.

```
docker inspect nombre_del_contenedor
```

Paso 8: Detener un Contenedor

Para detener un contenedor en ejecución, puedes utilizar el siguiente comando:

```
docker stop ID_del_contenedor
```

Reemplaza `ID_del_contenedor` con el ID real del contenedor que deseas detener.

Paso 9: Eliminar un Contenedor

Antes de continuar probemos el comando docker container prune, este comando elimina todos los contenedores que no estén en ejecución.

```
docker container prune
```

Para eliminar un contenedor que no está en ejecución, puedes utilizar el siguiente comando:

```
docker rm ID_del_contenedor
```

Reemplaza `ID_del_contenedor` con el ID real del contenedor que deseas eliminar.

1.6 Docker desde Visual Studio Code.

los pasos para instalar la extensión de Docker en Visual Studio Code y cómo utilizarla para trabajar con Docker en tu proyecto.

Paso 1: Instalar la extensión de Docker para Visual Studio Code

Abre Visual Studio Code.

Haz clic en el icono de extensiones en el panel lateral izquierdo o presiona Ctrl+Shift+X (o Cmd+Shift+X en macOS) para abrir la sección de extensiones.

En el cuadro de búsqueda, escribe “Docker” y selecciona la extensión “Docker” desarrollada por Microsoft.

Haz clic en el botón “Instalar” para instalar la extensión.

Una vez instalada, verás un nuevo icono de Docker en la barra lateral izquierda de VS Code.

Paso 2: Utilizar la extensión de Docker en tu proyecto

Una vez que la extensión de Docker está instalada, puedes utilizarla para gestionar imágenes y contenedores de Docker en tu proyecto.

Abre la carpeta de tu proyecto en Visual Studio Code.

Haz clic en el icono de Docker en la barra lateral izquierda para abrir la vista de Docker.

En la vista de Docker, verás diferentes secciones como “Imagenes”, “Contenedores”, “Volúmenes” y “Redes”.

Para construir una imagen de Docker para tu proyecto, haz clic con el botón derecho en la carpeta raíz de tu proyecto y selecciona “Build Image…” en el menú contextual. Esto abrirá una ventana donde puedes configurar los detalles de la imagen y su etiqueta.

Para crear y ejecutar un contenedor basado en la imagen que acabas de construir, haz clic con el botón derecho en la imagen en la sección “Imagenes” y selecciona “Run”. Esto abrirá una ventana donde puedes configurar las opciones del contenedor, como los puertos y las variables de entorno.

Una vez que el contenedor esté en ejecución, puedes ver sus registros y otros detalles haciendo clic en el contenedor en la sección “Contenedores”.

Puedes detener y eliminar contenedores desde la vista de Docker haciendo clic con el botón derecho en el contenedor y seleccionando “Stop” o “Remove”.

Estos son algunos de los pasos básicos para utilizar Docker desde Visual Studio Code con la extensión oficial de Docker.

La extensión proporciona una forma sencilla de gestionar imágenes y contenedores de Docker directamente desde el entorno de desarrollo de VS Code, lo que facilita el trabajo con Docker en tus proyectos.

Práctica: Dockerizar el Proyecto Django “Blog”

A continuación, vamos a dockerizar el proyecto Django **Blog** que hemos estado desarrollando. Para ello, vamos a seguir los siguientes pasos:

1. Crear un archivo `Dockerfile` en el directorio raíz del proyecto.

Este archivo contendrá las instrucciones para construir la imagen de Docker.

2. En el `Dockerfile`, especificar la imagen base que utilizaremos.

Por ejemplo, podemos usar la imagen oficial de Python para Django:

```
FROM python:3.8
```

3. Copiar el código fuente de nuestra aplicación al contenedor:

```
COPY . /app
```

4. Establecer el directorio de trabajo dentro del contenedor:

```
WORKDIR /app
```

5. Instalar las dependencias de Python necesarias para nuestro proyecto:

```
RUN pip install -r requirements.txt
```

6. Exponer el puerto en el que se ejecutará nuestro servidor de Django:

```
EXPOSE 8000
```

7. Especificar el comando para ejecutar nuestra aplicación:

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

8. Guardar y cerrar el archivo `Dockerfile`.
9. Desde la línea de comandos, navegar al directorio raíz del proyecto y ejecutar el siguiente comando para construir la imagen de Docker:

```
docker build -t nombre_de_la_imagen .
```

Reemplaza `nombre_de_la_imagen` con un nombre significativo para tu imagen.

10. Una vez que la imagen se haya construido correctamente, puedes crear un contenedor basado en esa imagen:

```
docker run -p 8000:8000 nombre_de_la_imagen
```

Reemplaza `nombre_de_la_imagen` con el nombre de la imagen que creaste en el paso anterior.

11. Ahora, deberías poder acceder a tu aplicación Django desde tu navegador ingresando `http://localhost:8000`.

Con estos pasos, habrás dockerizado exitosamente tu proyecto Django **Blog** y estará listo para ser desplegado en cualquier entorno que tenga Docker instalado.

1.7 Introducción a Docker Compose

Docker Compose es una herramienta que permite definir y gestionar aplicaciones multi-contenedor. Permite definir las dependencias, servicios, redes y volúmenes que componen una aplicación en un archivo YAML, lo que facilita la configuración y el despliegue de la aplicación en diferentes entornos.

A diferencia de Docker, que se enfoca en la construcción y ejecución de contenedores individuales, Docker Compose se centra en la orquestación de múltiples contenedores y su interacción.

1.8 Ventajas de usar Docker Compose

Definición clara de servicios: Con Docker Compose, podemos definir todos los servicios necesarios para nuestra aplicación en un solo archivo YAML, lo que facilita la gestión de la infraestructura de la aplicación.

Configuración simplificada: Docker Compose permite establecer todas las configuraciones necesarias para cada servicio en un solo lugar, lo que facilita la gestión de la configuración y el despliegue en diferentes entornos.

Interconexión de servicios: Docker Compose facilita la conexión de diferentes servicios y la comunicación entre ellos, lo que es especialmente útil para aplicaciones que requieren una arquitectura de microservicios.

Escalabilidad: Docker Compose permite escalar servicios individuales según las necesidades de la aplicación, lo que facilita el escalado horizontal de la aplicación.

1.8.1 Cómo usar Docker Compose en nuestro proyecto Django "Blog"

Para utilizar Docker Compose en nuestro proyecto Django **Blog**, seguiremos los siguientes pasos:

1. Crear un archivo docker-compose.yml en el directorio raíz del proyecto.
2. Este archivo contendrá la configuración de los servicios y contenedores de nuestra aplicación.

En el archivo docker-compose.yml, definir los servicios necesarios para nuestra aplicación. Por ejemplo, podemos definir dos servicios: uno para el servidor de Django y otro para la base de datos.

```yaml
version: '3'

services:
  db:
    image: postgres
    environment:
      POSTGRES_DB: blog
      POSTGRES_USER: myuser
      POSTGRES_PASSWORD: mypassword

  web:
    build: .
    command: python manage.py runserver 0.0.0.0:8000
```

```
    ports:
      - "8000:8000"
    depends_on:
      - db
```

Guardar y cerrar el archivo docker-compose.yml.

Desde la línea de comandos, navegar al directorio raíz del proyecto y ejecutar el siguiente comando para construir las imágenes de los servicios definidos en el archivo docker-compose.yml:

```
docker-compose build
```

Una vez que las imágenes se hayan construido correctamente, puedes ejecutar el siguiente comando para iniciar los contenedores definidos en el archivo docker-compose.yml:

```
docker-compose up
```

Ahora, deberías poder acceder a tu aplicación Django desde tu navegador ingresando http://localhost:8000.

Con estos pasos, habrás utilizado Docker Compose para gestionar los servicios y contenedores de tu aplicación Django "Blog". Docker Compose simplifica el despliegue y la gestión de aplicaciones multi-contenedor, lo que lo convierte en una herramienta muy útil para el desarrollo de aplicaciones complejas y escalables.