Crea tu página web desde cero en Docker

Vamos a montar una página web desde cero utilizando Docker, combinando un archivo `docker-compose.yml`, un `Dockerfile`, y una configuración personalizada de Nginx.

Requisitos previos

Antes de empezar, asegúrate de tener instalados los siguientes programas:

- **Docker**: para crear y gestionar contenedores.
- **Docker Compose**: para orquestar tus servicios.

¿Cómo instalarlos?

Puedes consultar la documentación oficial de Docker para guías detalladas de instalación:

- Instalar Docker
- Instalar Docker Compose

Si tu sistema está basado en **APT** (como Debian o Ubuntu), también puedes seguir este tutorial paso a paso:

Instalación y primeros pasos con Docker

También necesitarás los archivos de tu página web en un directorio local.

Ventajas de dockerizar tu página web

Dockerizar una página web ofrece una serie de beneficios importantes:

1. **Portabilidad**
   Tu aplicación funcionará de la misma manera en cualquier sistema que tenga Docker instalado, independientemente de las diferencias en el entorno.

2. **Consistencia entre entornos**
   Evita problemas típicos de configuración al garantizar que el entorno de desarrollo, pruebas y producción sean idénticos.
3. **Fácil despliegue**
   Con un simple comando (`docker compose up -d --build`), puedes levantar la infraestructura necesaria para tu aplicación sin complicaciones.
4. **Aislamiento**
   Cada contenedor opera de forma independiente, evitando conflictos con otros servicios o aplicaciones en el mismo sistema.
5. **Ligereza**
   Los contenedores consumen menos recursos que las máquinas virtuales, ya que comparten el kernel del sistema operativo.
6. **Escalabilidad**
   Permite escalar horizontalmente al replicar contenedores según las necesidades de tráfico.
7. **Gestión de versiones**
   Facilita la gestión de versiones al permitir etiquetar diferentes estados de la aplicación y realizar rollbacks en caso necesario.
8. **Integración con CI/CD**
   Docker se integra fácilmente con herramientas de integración continua, automatizando pruebas y despliegues.
9. **Simplificación del manejo de dependencias**
   No necesitas instalar dependencias o configuraciones específicas en tu máquina local; todo está definido en el `Dockerfile`.
10. **Facilidad para pruebas locales**
    Permite realizar pruebas en un entorno controlado sin afectar otros servicios.
11. **Ecosistema rico**
    Docker ofrece acceso a imágenes preconstruidas en Docker Hub, como servidores web o bases de datos, para agilizar configuraciones.
12. **Resiliencia**
    Configuraciones como `restart: unless-stopped` aseguran que el contenedor se reinicie automáticamente en caso de fallos.
13. **Compatibilidad futura**
    Docker es una tecnología moderna y ampliamente adoptada, asegurando relevancia a largo plazo.

Paso 1: Preparar el archivo `docker-compose.yml`

Este archivo define cómo se orquestará tu contenedor web. Aquí tienes un ejemplo:

```
services:
  web: # Define un servicio llamado "web". Este será el contenedor que ejecutará tu página web.
    build: . # Indica que se construirá una imagen Docker utilizando el Dockerfile que está en el directorio actual.
    network_mode: bridge # Configura el contenedor para usar el modo de red "bridge". Este es el modo predeterminado, que aísla la red del contenedor y la conecta a través de un puente virtual al host.
    volumes: # Define volúmenes que conectan el sistema de archivos del host con el contenedor.
      - ./website:/usr/share/nginx/html # Monta la carpeta local "./website" (donde está tu página web) en "/usr/share/nginx/html" dentro del contenedor, para que Nginx pueda servir los archivos.
      - /etc/localtime:/etc/localtime:ro # Sincroniza la hora local del host con el contenedor para que compartan la misma zona horaria. Se monta en modo "ro" (solo lectura) para evitar cambios accidentales.
    ports:
      - "8080:80" # Mapea el puerto 80 del contenedor (donde escucha Nginx) al puerto 8080 del host, haciendo accesible la aplicación en "http://localhost:8080".
    restart: unless-stopped # Configura el contenedor para reiniciarse automáticamente a menos que sea detenido manualmente. Esto asegura alta disponibilidad en caso de reinicios o fallos.
    container_name: my_web_service # Asigna un nombre personalizado al contenedor, facilitando su identificación en comandos como "docker ps".
    environment: # Define variables de entorno que se pasarán al contenedor.
      - TZ=Europe/Madrid # Configura la zona horaria dentro del contenedor para que coincida con la local (Europa/Madrid en este caso).
```

Paso 2: Crear el `Dockerfile`

El `Dockerfile` define cómo se construirá la imagen de tu contenedor. Aquí tienes un ejemplo:

```
# Usa una imagen base ligera de servidor web
FROM nginx:alpine

# Copia los archivos de la página al directorio de Nginx
COPY ./website /usr/share/nginx/html
```

```
# Copia el archivo de configuración personalizado de Nginx
COPY nginx.conf /etc/nginx/nginx.conf

# Exponer el puerto 80 para que se pueda acceder desde fuera del contenedor
EXPOSE 80
```

Paso 3: Configurar Nginx

Aquí tienes un ejemplo de un archivo `nginx.conf` personalizado, adaptado para este proyecto:

```
worker_processes 1;

events {
    worker_connections 1024;
}

http {
    include       mime.types;
    default_type  application/octet-stream;

    client_max_body_size 10M;

    limit_req_zone $binary_remote_addr zone=mylimit:10m rate=20r/m;

    server {
        listen 80;
        server_name localhost;

        root /usr/share/nginx/html;
        index index.html;

        error_page 400 401 403 404 500 502 503 504 /error.html;
        location = /error.html {
            root /usr/share/nginx/html;
            internal;
        }

        location / {
            limit_req zone=mylimit burst=5 nodelay;
```

```
            try_files $uri $uri/ =404;
        }

        location /resources/ {
            limit_req zone=mylimit burst=20 delay=5;
            try_files $uri $uri/ =404;
        }

        location ~ /\.ht { deny all; }
        location ~* \.(conf|ini|log|yml|)$ { deny all; }
        location ~ /\. { deny all; }
        location ~* /(admin|dashboard) { deny all; }
    }
}
```

Paso 4: Levantar tu entorno

Con los archivos listos (`docker-compose.yml`, `Dockerfile`, `nginx.conf`, y tus archivos web en el directorio `website`), ejecuta el siguiente comando:

```
docker compose up -d --build
```

Este comando construye la imagen, levanta el contenedor en segundo plano y actualiza el entorno si ya estaba corriendo.

Exponer tu página al exterior

Si deseas que tu página sea accesible desde Internet, puedes exponerla utilizando un proxy inverso como Caddy. Sigue esta guía de Caddy para gestionar dominios de forma sencilla, con soporte para HTTPS incluido.

Verificar el contenedor

Para asegurarte de que todo funciona correctamente, ejecuta:

```
docker ps
```

Deberías ver tu contenedor `my_web_service` en ejecución, con el puerto 8080 expuesto.

Notas finales

Este setup es ideal para entornos de desarrollo. Si planeas usarlo en producción, considera:

- Utilizar HTTPS para conexiones seguras.
- Añadir reglas de seguridad avanzadas en Nginx.
- Configurar una solución para gestionar los logs de tu contenedor.

Revision #3
Created 2024-12-05 12:58:46 UTC by Juan Francisco
Updated 2026-01-21 12:57:21 UTC by Juan Francisco