Curso Docker

Arturo Silvelo

Try New Roads

Tabla de Contenido

1. Introducción

2. Contenedores

3. Redes

4. Volúmenes

5. Imágenes

6. Docker Compose

Introducción

Contenedores

Redes

¿Qué es una Red?

- **Red**: Conjunto de dispositivos conectados entre sí para compartir recursos y comunicarse.
- **Ejemplo**: La red de Wi-Fi en casa permite a los dispositivos conectarse a Internet y entre ellos.

¿Qué es una Dirección IP?

- **Dirección IP**: Identificador único asignado a cada dispositivo en una red.
- **Formato**: Una dirección IP en formato IPv4 se compone de 4 octetos (ej. 192.168.1.1).
- **Función**: Permite identificar y localizar dispositivos dentro de una red.

Tipos de IP - Públicas y Privadas

- **IP Pública**: Visible en Internet y asignada por el proveedor de Internet (ISP).
- **IP Privada**: Solo se usa dentro de redes locales; no es accesible desde Internet.

Clases de IP (A, B, C)

- **Clase A**: Rango de direcciones de 0.0.0.0 a 127.255.255.255.
  - **IP Privada Clase A**: 10.0.0.0 - 10.255.255.255
- **Clase B**: Rango de direcciones de 128.0.0.0 a 191.255.255.255.
  - **IP Privada Clase B**: 172.16.0.0 - 172.31.255.255
- **Clase C**: Rango de direcciones de 192.0.0.0 a 223.255.255.255.
  - **IP Privada Clase C**: 192.168.0.0 - 192.168.255.255

Máscaras de Red

- **Máscara de Red**: Define qué parte de la IP identifica la red y cuál el dispositivo.
- **Ejemplo**: Máscara 255.255.255.0 indica que los primeros 3 octetos son la red, el último es el dispositivo.
- **Subredes**: Permiten dividir una red grande en redes más pequeñas.

Ejemplo Práctico de Dirección IP y Máscara de Red (Clase B)

Ejemplo de Configuración de Red:

- **Dirección IP**: 172.16.10.25
- **Máscara de Red**: 255.255.0.0 ó 172.16.10.25/16 (11111111.11111111.00000000.00000000)

Explicación de los Componentes:

- **Dirección IP**: Identifica a un dispositivo específico dentro de la red.
- **Máscara de Red**: Los primeros 16 bits (172.16) representan la red.
- **Notación CIDR**: En notación CIDR, la máscara de red 255.255.0.0 se representa como /16, lo que indica que los primeros 16 bits están reservados para la red y los 16 bits restantes para los hosts.
- **Rango de la Red**: Desde 172.16.0.1 hasta 172.16.255.254.

Ejemplo Práctico de Dirección IP y Máscara de Red (Clase C) con Notación CIDR

Ejemplo de Configuración de Red:

- **Dirección IP**: 192.168.1.10
- **Máscara de Red**: 255.255.255.0 ó 192.168.1.10/24 (11111111.11111111.11111111.00000000)

Explicación de los Componentes:

- **Dirección IP**: Identifica a un dispositivo específico dentro de la red.
- **Máscara de Red**: Los primeros 24 bits (192.168.1) representan la red.
- **Notación CIDR**: En notación CIDR, la máscara de red 255.255.255.0 se representa como /24, lo que indica que los primeros 24 bits están reservados para la red y los 8 bits restantes para los hosts.
- **Red**: La dirección de red es 192.168.1.0, que es el identificador de la red.
- **Rango de la Red**: Desde 192.168.1.1 hasta 192.168.1.254.

Introducción a Redes en Docker

- Las redes en Docker permiten la comunicación entre contenedores.
- Proporcionan aislamiento y control sobre cómo se comunican los contenedores.
- Las redes pueden persistir más allá de la vida de los contenedores.

Tipos de Redes en Docker (Parte 1)

bridge (predeterminada):

- Red privada para los contenedores que se ejecutan en el mismo host.
- Ideal para entornos de desarrollo donde los contenedores necesitan comunicarse entre sí.
- Los contenedores pueden acceder al exterior a través del gateway, pero están aislados de otros contenedores.

Tipos de Redes en Docker (Parte 1)

- El rango de IPs predeterminado para la red ‘bridge‘ es `172.17.0.0/16`.
- Docker automáticamente asigna una IP dentro de este rango cuando un contenedor se ejecuta en esta red.
- Modificar el rango por defecto del bridge `https://docs.docker.com/engine/network/drivers/bridge`

Tipos de Redes en Docker (Parte 1)

Comandos de Ejemplo:

```
docker network create --subnet 172.20.0.0/16
    my_network

docker network create  --driver bridge --subnet
    172.19.0.0/16 my_network_2

docker network create my_network_3

docker network create --subnet 172.20.0.0/16
    my_network_4

docker network inspect <NETWORK_NAME|NETWORK_ID>
```

Tipos de Redes en Docker (Parte 2)

host:

- El contenedor comparte la red del host, eliminando la capa de aislamiento.
- Utiliza la red del sistema directamente, mejorando el rendimiento en aplicaciones que requieren baja latencia.
- Sin embargo, esto sacrifica el aislamiento entre contenedores.

Comando de Ejemplo:

```
docker run --network host -d nginx
```

Tipos de Redes en Docker (Parte 3)

none:

- No se asigna ninguna red al contenedor, dejándolo completamente aislado.
- Útil para pruebas de seguridad o para aplicaciones que no necesitan comunicación de red.

Comando de Ejemplo:

```
docker run --network none -d busybox top
docker exec -it <container_id> ping google.com
```

Comparación de Tipos de Redes en Docker

Características	Aislamiento	Acceso a la Red Externa	Usos Comunes
bridge	Aislado entre contenedores y host.	A través de NAT y gateway.	Desarrollo local, múltiples contenedores en un mismo host.
host	No hay aislamiento, comparte la red del host.	Directo, sin NAT.	Contenedores de alto rendimiento, aplicaciones que requieren baja latencia.
none	Totalmente aislado, sin acceso a la red.	No tiene acceso a la red.	Pruebas de seguridad, contenedores que no necesitan conectividad.

Ejercicio: Conectar dos aplicaciones utilizando una red en Docker (Parte 1)

Enunciado:

Tienes dos imágenes Docker disponibles en Docker Hub:

- `silvelo/todo-backend`
- `silvelo/todo-client`

El objetivo de este ejercicio es desplegar ambos contenedores y configurarlos para que se comuniquen entre sí utilizando una red personalizada en Docker.

1. Crea una red personalizada llamada `todo-network` que permita la comunicación entre los contenedores.
2. Inicia un contenedor basado en la imagen `silvelo/todo-backend` y conéctalo a la red `todo-network`.
3. Asegúrate de exponer el puerto `3000` para el backend.

Ejercicio: Conectar dos aplicaciones utilizando una red en Docker (Parte 2)

4. Inicia un contenedor basado en la imagen `silvelo/todo-client` y conéctalo también a la red `todo-network`.
5. Asegúrate de que la aplicación frontend pueda acceder al backend a través del nombre del servicio (`todo-backend`) en lugar de una dirección IP.

Resultado esperado:

- El backend (`silvelo/todo-backend`) debería estar accesible en la red bajo el nombre `todo-backend`.
- El frontend (`silvelo/todo-client`) debería poder comunicarse con el backend usando la URL `http://todo-backend:3000`.

Ejercicio: Conectar aplicaciones y base de datos (Parte 3)

Extensión del ejercicio:

Vamos a añadir una base de datos MongoDB y configurar la red para que los servicios tengan permisos específicos.

1. Añade un contenedor basado en la imagen oficial de `mongo`.
2. El backend debe poder conectarse tanto al cliente como a MongoDB.
3. El cliente solo puede conectarse al backend, pero no directamente a MongoDB.
4. MongoDB solo debe ser accesible desde el backend.

Volúmenes

¿Qué son los volúmenes en Docker?

Los **volúmenes** en Docker son un mecanismo para almacenar y persistir datos.

- **Persistir datos**: Los datos no se pierden al detener o eliminar un contenedor.
- **Compartir datos entre contenedores**: Permiten que varios contenedores accedan al mismo conjunto de datos.

Independientes del contenedor: Los volúmenes existen de forma separada, por lo que los datos permanecen incluso si el contenedor se elimina.

Ejemplo práctico: Persistencia de datos en MongoDB

- Si ejecutamos un contenedor de MongoDB sin un volumen, **todos los datos se perderán** si el contenedor se elimina.
- Al usar un volumen, los datos se almacenan de forma persistente en el sistema del host.

```
docker run -d -p 27017:27017 --name mongodb -v
    mongodb_data:/data/db mongo
```

- En este ejemplo, el volumen `mongodb_data` almacena los datos persistentes de MongoDB.
- Incluso si el contenedor se elimina, los datos quedan guardados en el volumen y pueden reutilizarse en un nuevo contenedor.

Tipos de Volúmenes en Docker

- Volúmenes gestionados por Docker (Named Volumes)
- Montajes de directorios del host (Bind Mounts)
- Volúmenes temporales (Anonymous Volumes)

Volúmenes gestionados por Docker (Named Volumes)

- Son volúmenes creados y gestionados automáticamente por Docker, sin necesidad de configurarlos manualmente.
- Se almacenan en una ubicación predeterminada dentro del sistema de archivos del host (generalmente en `/var/lib/docker/volumes`).
- Docker asegura la persistencia de los datos, incluso si el contenedor se detiene o elimina.
- **Ventaja principal:** Los datos persisten independientemente del ciclo de vida del contenedor.
- **Uso común:** Almacenar bases de datos y otros datos que deben mantenerse a través de múltiples ciclos de vida de contenedores.

```
docker volume create volumen_nombre
docker run -d -v volumen_nombre:/data/db mongo
```

Montajes de directorios del host (Bind Mounts)

- Los **bind mounts** permiten usar directorios del host directamente dentro del contenedor.
- El contenedor puede **leer y escribir** en estos directorios, permitiendo acceso directo a los archivos del host.
- **Ventajas:**
  - Permite compartir archivos entre el host y el contenedor, útil para logs, configuraciones, bases de datos, etc.
  - Cambios realizados en el host se reflejan inmediatamente en el contenedor y viceversa.
- **Desventajas:**
  - Riesgo de corrupción de datos si el contenedor y el host no están bien sincronizados.
  - Dependencia de la estructura del sistema de archivos del host.
- **Uso común:** En entornos de desarrollo, donde se necesitan cambios inmediatos entre el código del host y el contenedor.

```
docker run -d -v ${PWD}/index.html:/usr/share/
    nginx/html/index.html nginx
```

Volúmenes temporales (Anonymous Volumes)

- Volúmenes creados automáticamente por Docker sin un nombre explícito.
- Estos volúmenes no tienen un nombre y **persisten** después de que el contenedor se elimina, aunque no se pueden acceder fácilmente sin un nombre.
- **Ventajas:** Útiles para almacenar datos temporales generados por un contenedor, como archivos de logs o datos temporales.
- **Desventajas:** No se pueden gestionar fácilmente y pueden acumularse si no se eliminan explícitamente.
- **Uso común:** Datos que solo deben existir durante la vida del contenedor, como archivos temporales generados durante su ejecución.

```
docker run -d -v /data/db mongo
```

Ejercicio: Persistencia de Datos con Volúmenes

- Partiendo del ejercicio de redes donde se conectan las aplicaciones `todo-backend` y `todo-client`, realiza lo siguiente:
- Crea un volumen nombrado para almacenar la base de datos de MongoDB.
- Inserta algunas entradas de ejemplo en la base de datos de MongoDB a través del backend.
- Detén y elimina el contenedor de MongoDB.
- Crea un nuevo contenedor de MongoDB usando el mismo volumen nombrado.
- Verifica que los datos insertados anteriormente siguen estando presentes en la base de datos.

Ejercicio 1: Persistencia de Datos con Volúmenes

Partiendo del ejercicio de redes donde se conectan las aplicaciones `todo-backend` y `todo-client`, realiza lo siguiente:

1. Crea un volumen nombrado para almacenar la base de datos de MongoDB y añade dicho volumen al contenedor.
2. Inserta algunas entradas de ejemplo en la base de datos de MongoDB a través del backend.
3. Elimina el contenedor de MongoDB.
4. Crea un nuevo contenedor de MongoDB usando el mismo volumen nombrado.
5. Verifica que los datos insertados anteriormente siguen estando presentes en la base de datos.

Ejercicio 2: Persistencia de Datos con Volúmenes

Partiendo del ejercicio anterior:

1. Haz una copia del volumen en un directorio local.
2. Creamos una nueva máquina de Mongo con un volumen de tipo `bind`.
3. Verifica que ambas bases de datos tengan los mismos datos.

Ejercicio 3: Persistencia de Datos con Volúmenes

1. Crea una nueva instancia de Mongo con un volumen anónimo e inspecciona la información de los volúmenes.
2. Inserta algunas entradas de ejemplo en la base de datos de MongoDB.
3. Detén y borra el contenedor.
4. Recupera los datos del volumen e intenta montarlos en otro contenedor.

Imágenes

Docker Compose