
# Curso Docker

Arturo Silvelo

Try New Roads

# Tabla de Contenido

# 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 generados por y utilizados por contenedores.
- A diferencia de los contenedores, los volúmenes existen de forma independiente y pueden ser compartidos entre contenedores.
- Los volúmenes se pueden crear y gestionar desde Docker y no dependen del ciclo de vida de los contenedores.
- Se almacenan en el sistema de archivos del host, lo que permite que los datos persistan incluso si el contenedor se detiene o se elimina.

# ¿Por qué son necesarios los volúmenes?

- Los volúmenes son necesarios porque los contenedores de Docker son efímeros: **se eliminan con facilidad**.
- Sin volúmenes, los datos dentro de los contenedores se perderían cuando estos se reinician o se eliminan.
- Los volúmenes permiten que los datos se conserven de manera persistente, incluso si los contenedores se recrean.
- Son cruciales cuando se quiere compartir datos entre varios contenedores o almacenar configuraciones y bases de datos.

# Ejemplo con MongoDB

- Imagina que estamos ejecutando un contenedor de MongoDB sin volumen.
- Si el contenedor se detiene o se elimina, **todos los datos de la base de datos se perderán**.
- Para evitar esto, usamos un volumen para persistir la base de datos.
- Ejemplo de comando para ejecutar MongoDB con un volumen:

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

- En este ejemplo, el volumen `mongodb_data` guarda los datos persistentes de MongoDB en el contenedor.
- Incluso si el contenedor de MongoDB se elimina, los datos siguen estando disponibles en el volumen.

# 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 por Docker, sin necesidad de configurarlos manualmente.
- Se almacenan en una ubicación predeterminada dentro del sistema de archivos del host.
- Docker maneja la persistencia de los datos, incluso si el contenedor se detiene o elimina.
- **Ventaja principal:** Los datos se mantienen aunque se elimine el contenedor.
- **Uso común:** Base de datos y otros datos que deben persistir más allá del ciclo de vida del contenedor.

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

# Montajes de directorios del host (Bind Mounts)

- Montajes que permiten usar directorios del host dentro del contenedor.
- El contenedor puede leer y escribir en estos directorios.
- **Ventajas:** Compartir archivos entre el host y el contenedor (logs, configuraciones, etc.).
- **Desventajas:** Riesgo de corrupción de datos si no se configura correctamente.
- **Uso común:** Desarrollo, donde los cambios deben reflejarse inmediatamente.

```
docker run -d -v ${PWD}/mongo-db:/data/db mongo
```

# Volúmenes temporales (Anonymous Volumes)

- Volúmenes creados automáticamente por Docker sin un nombre.
- No se puede acceder a ellos después de que el contenedor se elimine.
- **Ventajas:** Útiles para almacenar datos temporales generados por un contenedor.
- **Desventajas:** No permiten la persistencia a largo plazo.
- **Uso común:** Datos que solo deben existir durante la vida del contenedor.

```
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, pero mantén el volumen persistente.
- 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.

# Imágenes

# Docker Compose