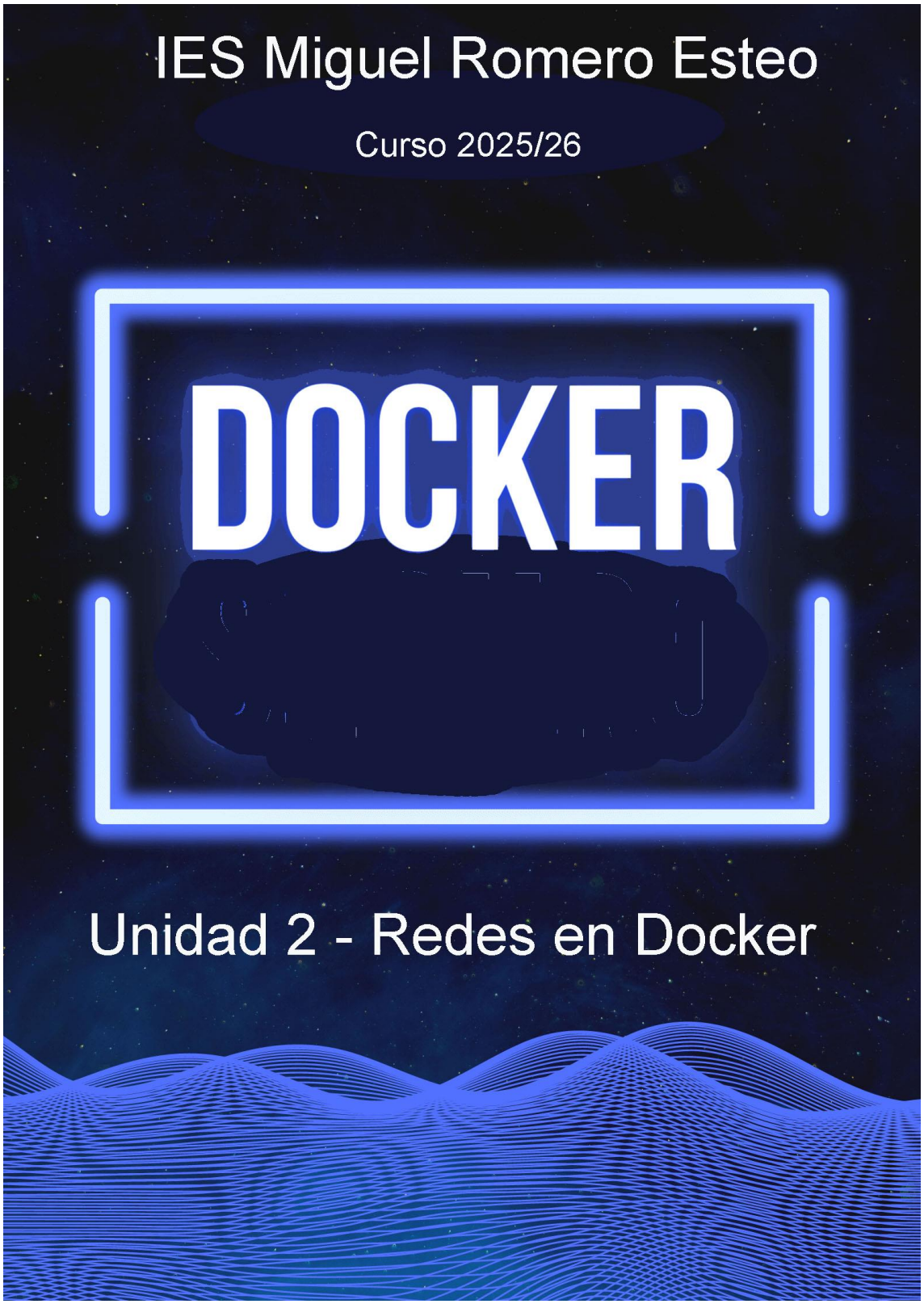
IES Miguel Romero Esteo

Curso 2025/26



Unidad 2 - Redes en Docker

UNIDAD 2 – REDES EN DOCKER

ÍNDICE

1.- Tipos de redes en Docker

Docker no solo permite ejecutar aplicaciones en contenedores, también provee un sistema de redes virtuales que facilita la comunicación entre ellos y con el exterior.
Las redes en Docker abstraen la conectividad y permiten controlar cómo se comunican los contenedores entre sí y con la máquina host.

1. Bridge (puente)

Es la red por defecto que Docker crea (bridge).
Los contenedores en esta red pueden comunicarse entre ellos usando su IP interna.
Permite exponer puertos al host con -p.

Ejemplo:

```
docker run -d --name web1 --network bridge -p 8080:80 nginx
```

☞ Acceso desde host: http://localhost:8080

2. Host

El contenedor comparte directamente la red del host. No hay aislamiento de red entre contenedor y host. Útil cuando quieres que el contenedor use la misma IP del host.

Ejemplo:

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

☞ El contenedor escucha en el mismo puerto que el host.

3. None

El contenedor no tiene conectividad de red. Solo puede comunicarse a través de volúmenes u otros mecanismos (no red). Útil para contenedores que no requieren red o por seguridad.

Ejemplo:

docker run -it --network none alpine sh

4. User-defined (redes personalizadas)

El usuario puede crear redes personalizadas (bridge, overlay, etc.). Proveen aislamiento, resolución de nombres y comunicación entre contenedores. Muy útil para aplicaciones multi-contenedor (por ejemplo, con Docker Compose).

Ejemplo:

docker network create mi_red
docker run -d --name db --network mi_red mysql
docker run -d --name app --network mi_red python-app

☞ Aquí app puede conectarse a db usando mysql:3306.

5. Overlay

Redes distribuidas que permiten comunicación entre contenedores en diferentes hosts dentro de un Swarm. Usadas en entornos de orquestación (Docker Swarm, Kubernetes).

2.- **Comandos útiles de redes en Docker**

- Listar redes: docker network ls

docker network ls

NETWORK ID	NAME	DRIVER	SCOPE
3102da5c36a4	bridge	bridge	local
ab7a35c936b8	host	host	local
ce90912da876	mi-red	bridge	local
71893eecbc77	none	null	local
60a525c86ddb	nuevared	bridge	local

- Inspeccionar una red: docker network inspect bridge

docker network inspect bridge

```
[
   {
      "Name": "bridge",
      "Id":
"3102da5c36a4929cd2a468991c877f85998da380e42d729b2ce7db737179a96
4",
      "Created": "2025-08-25T11:55:47.4952485Z",
      "Scope": "local",
      "Driver": "bridge",
      "EnableIPv6": false,
      "IPAM": {
         "Driver": "default",
         "Options": null,
```


- Crear red personalizada: docker network create mi_red

docker network create nueva_red
d6f426ebec27e6399963c898af902709d9ac7daea6188be423e0ca2e3773224e

Devuelve un has que identifica a la red. Podemos ver que se ha creado con
PS C:\docker\alpine> docker network ls

NETWORK ID	NAME	DRIVER	SCOPE
3102da5c36a4	bridge	bridge	local
ab7a35c936b8	host	host	local
ce90912da876	mi-red	bridge	local

- Conectar contenedor a una red:

docker network connect nueva_red mi_alpine

donde nueva-red es la red creada anteriormente y mi_alpine es un contenedor.

- Desconectar contenedor de una red:

docker network disconnect nueva_red mi_alpine

3.- **Comunicación entre contenedores**

En redes personalizadas (user-defined bridge), Docker asigna un DNS interno a cada contenedor.

Ejemplo: app puede conectarse a db simplemente usando el nombre db.

En la red por defecto (bridge), la comunicación solo funciona por IP, a menos que se configure manualmente.

Para que un contenedor sea accesible desde fuera (host u otros equipos), se deben mapear puertos con -p:

docker run -d -p 8080:80 nginx

☞ El puerto 80 del contenedor queda expuesto en el puerto 8080 del host.