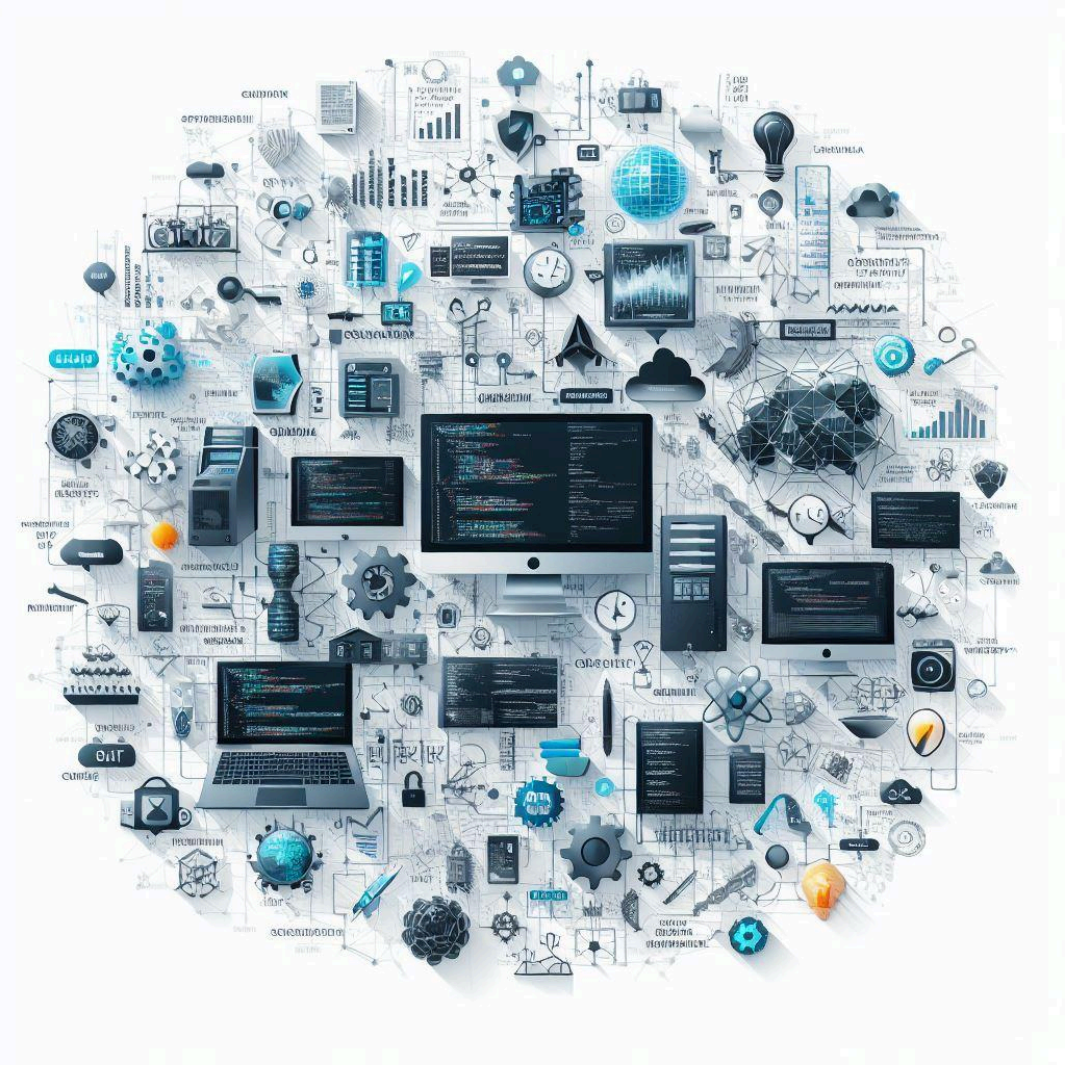


Sistemas Informáticos

Desarrollo de Aplicaciones Web
Desarrollo de Aplicaciones Multiplataforma
CURSO 24-25

Guillermo Domínguez de la Fuente
I.E.S Valle del Jerte (Plasencia)
Sistemas Informáticos
Curso 24-25

Actividad práctica 4

Ejercicio Práctico - Subnetting con Docker

La empresa **DockerTech** dispone del bloque de red **172.30.100.0/24** para alojar sus microservicios Docker en un entorno de red segmentado. Se necesita dividir esta red en **subredes independientes**, asignando a cada una un tamaño adecuado en función de la **cantidad de contenedores Docker** que requieren. Para optimizar el uso del espacio de direcciones IP, deberás aplicar **Subnetting con VLSM (Variable Length Subnet Masking).**

A continuación, se detallan los servicios y la **cantidad estimada de contenedores Docker** que cada uno necesita:

- **Servicio Frontend:** 50 contenedores
- **Servicio Backend:** 30 contenedores
- **Servicio Base de Datos:** 12 contenedores
- **Servicio de Cache:**6 contenedores
- **Servicio de Monitoreo:** 1 contenedores

Objetivo

Realiza el subnetting aplicando VLSM para asignar bloques de direcciones IP a cada servicio, de acuerdo con sus necesidades específicas. Asegúrate de que **no se desperdicien direcciones IP** innecesariamente.

Para cada subred generada, deberás indicar:

1. Nombre del servicio asignado
2. Nueva máscara de subred (en notación decimal)
3. Dirección de red (Network Address)
4. Primera dirección IP válida (para hosts)
5. Última dirección IP válida (para hosts)
6. Dirección de broadcast
7. Número máximo de contenedores Docker (hosts) posibles en esa subred

docker network create --driver bridge --subnet 172.30.100.0/26 frontend_net

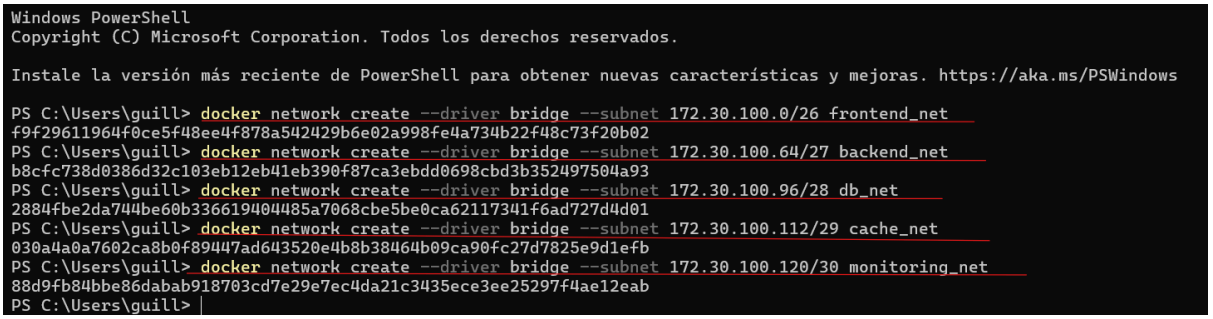
docker network create --driver bridge --subnet 172.30.100.64/27 backend_net

docker network create --driver bridge --subnet 172.30.100.96/28 db_net

docker network create --driver bridge --subnet 172.30.100.112/29 cache_net

docker network create --driver bridge --subnet 172.30.100.120/30 monitoring_net

Nota: Usamos ***–driver bridge*** para asegurarnos de que cada departamento esté bien contenido dentro de su propia red.



A continuación mostraremos las máscaras de subred, las direcciones de red, el rango de IP's válidas y el máximo de hosts por contenedor:

Servicio	Máscara de Subred (Decimal)	Dirección de Red	Primera IP válida	Última IP válida	Dirección de Broadcast	Nº máximo de contenedores (hosts)
Frontend (50 hosts)	/26	172.30.100.0	1	62	63	50
Backend (30 hosts)	/27	172.30.100.64	65	94	95	30
Base de Datos (12)	/28	172.30./100.96	97	110	111	12
Cache (8)	/29	172.30.100.112	113	118	119	6
Monitoreo (4)	/30	172.30.100.120	121	122	123	1

Ahora vamos a asignar una imagen distinta a cada contenedor (no se ha especificado acerca de esto en el enunciado del ejercicio, por lo que las imágenes utilizadas son las primeras que hemos encontrado sin seguir ninguna clase de criterio. Especialmente en el caso de alpine sleep infinity, puesto que es una imagen utilizada para testear y no hace nada: tal como su propio nombre indica, se pone a dormir para siempre.

docker run -dit --name frontend1 --network frontend_net nginx

```
PS C:\Users\guill> docker run -dit --name frontend1 --network frontend_net nginx
```

docker run -dit --name backend1 --network backend_net httpd

```
PS C:\Users\guill> docker run -dit --name backend1 --network backend_net httpd
```

docker run -dit --name db1 --network db_net mysql:5.7

```
PS C:\Users\guill> docker run -dit --name db1 --network db_net mysql:5.7
```

docker run -dit --name cache1 --network cache_net redis

```
PS C:\Users\guill> docker run -dit --name cache1 --network cache_net redis
```

docker run -dit --name monitor1 --network monitoring_net alpine sleep infinity

```
PS C:\Users\guill> docker run -dit --name monitor1 --network monitoring_net alpine sleep infinity
```

Usamos ***docker network ls*** para comprobar que nuestras redes han sido creadas correctamente:

```
PS C:\Users\guill> docker network ls
NETWORK ID     NAME                DRIVER    SCOPE
b5addb368eeb   \                   bridge    local
b8cfc738d038   backend_net         bridge    local
abeea98e3ab3   bridge              bridge    local
030a4a0a7602   cache_net           bridge    local
2884fbe2da74   db_net              bridge    local
f9f29611964f   frontend_net        bridge    local
d800a4d5da82   host                host      local
88d9fb84bbe8   monitoring_net      bridge    local
c92bce3ac220   none                null      local
55295ecdb52c   red_don_Guillermo   bridge    local
```