A continuación se muestra una lista de los comandos que se pueden usar con Docker networking:

```
$ docker network create
$ docker network connect
$ docker network ls
$ docker network rm
$ docker network disconnect
$ docker network inspect
```

docker network inspect

Este comando te permite saber los recursos empleados por una network así como su configuración

```
$ docker network inspect bridge
[
   {
       "Name": "bridge",
       "Id": "f7ab26d71dbd6f557852c7156ae0574bbf62c42f539b50c8ebde0f728a253b6f",
       "Scope": "local",
       "Driver": "bridge",
       "IPAM": {
           "Driver": "default",
           "Config": [
               {
                   "Subnet": "172.17.0.1/16",
                   "Gateway": "172.17.0.1"
               }
           ]
       },
       "Containers": {},
       "Options": {
           "com.docker.network.bridge.default_bridge": "true",
           "com.docker.network.bridge.enable_icc": "true",
           "com.docker.network.bridge.enable_ip_masquerade": "true",
           "com.docker.network.bridge.host_binding_ipv4": "0.0.0.0",
           "com.docker.network.bridge.name": "docker0",
           "com.docker.network.driver.mtu": "9001"
       }
   }
]
```

Ahora vemos el estado después de haber creado dos contenedores asociados a esa network:

```
$ docker network inspect bridge
{[
    {
        "Name": "bridge",
        "Id": "f7ab26d71dbd6f557852c7156ae0574bbf62c42f539b50c8ebde0f728a253b6f",
        "Scope": "local",
        "Driver": "bridge",
        "IPAM": {
            "Driver": "default",
            "Config": [
                {
                    "Subnet": "172.17.0.1/16",
                    "Gateway": "172.17.0.1"
                }
            ]
        },
        "Containers": {
            "3386a527aa08b37ea9232cbcace2d2458d49f44bb05a6b775fba7ddd40d8f92c": {
                "EndpointID": "647c12443e91faf0fd508b6edfe59c30b642abb60dfab890b4bdccee38750bc1",
                "MacAddress": "02:42:ac:11:00:02",
                "IPv4Address": "172.17.0.2/16",
                "IPv6Address": ""
            },
            "94447ca479852d29aeddca75c28f7104df3c3196d7b6d83061879e339946805c": {
                "EndpointID": "b047d090f446ac49747d3c37d63e4307be745876db7f0ceef7b311cbba615f48",
                "MacAddress": "02:42:ac:11:00:03",
                "IPv4Address": "172.17.0.3/16",
                "IPv6Address": ""
```

```
            }
        },
        "Options": {
            "com.docker.network.bridge.default_bridge": "true",
            "com.docker.network.bridge.enable_icc": "true",
            "com.docker.network.bridge.enable_ip_masquerade": "true",
            "com.docker.network.bridge.host_binding_ipv4": "0.0.0.0",
            "com.docker.network.bridge.name": "docker0",
            "com.docker.network.driver.mtu": "9001"
        }
    }
]
```

Contenedores en la red por defecto pueden comunicarse entre ellos usando diferentes IPS siempre y cuando la configuración de Docker lo soporte.

Nota: Docker no ofrece soporte para un service discovery automático cuando usamos la network bridge, sí quieres que tus contenedores puedan ser accesibles unos a otros debes de hacer uso de los Docker links.

docker network ls

Muestra un listado de las networks que Docker tiene creadas (por defecto, o no), inicialmente son tres.

```
$ docker network ls
NETWORK ID          NAME                DRIVER
9f104ee27bf5        none                null
cf23ee007fb4        host                host
7fsa4eb8c647        bridge              bridge
```

docker network create

Te permite crear tus propias networks: bridge o overlay. También es posible crear tus propias networks or plugins de red. Contenedores pueden comunicarse dentro de su network pero no a través de networks. Contenedores puede pertenecer a más de una red.

Vamos a crear una docker network de tipo bridge:

```
$ docker network create --driver bridge webvinar-docker
c5ee82f76de30319c75554a57164c682e7372d2c694fec41e42ac3b77e570f6b
```

```
$ docker network inspect webvinar-docker
[
    {
        "Name": "webvinar-docker",
        "Id": "c5ee82f76de30319c75554a57164c682e7372d2c694fec41e42ac3b77e570f6b",
        "Scope": "local",
        "Driver": "bridge",
        "IPAM": {
            "Driver": "default",
            "Config": [
                {}
            ]
        },
        "Containers": {},
        "Options": {}
    }
]
```

```
$ docker network ls
NETWORK ID          NAME                DRIVER
9f104ee27bf5        none                null
cf23ee007fb4        host                host
7fsa4eb8c647        bridge              bridge
```

```
c5ee82f76de3        webvinar-docker     bridge
```

Una vez la red fue creada correctamente podemos empezar a utilizar usando el parámetro `--net` .

```
$ docker run --net=webvinar-docker -itd --name=test busybox
885b7b4f792bae534416c95caa35ba123f201fa181e18e59beba0c80d7d77c1d
```

Cuando creas una network esta recibe una subnet que no ha sido utilizada todavía. Sin embargo, también es posible definir esa subnet usando el parámetro `--subnetwork` . Cuidado con usar la misma subnet para diferentes networks.

En una bridge network puedes solo una subnet mientras que en overlay networks puedes crear varias. Otro parámetro que se puede utilizar es `--gateway-ip-range` y `--aux-address`

```
$ docker network inspect webvinar-docker
```

Los contenedores dentro de esta red se podran comunicar entre sí e asimismo ellos estarán aislados del resto de networks. Para permitir a los contenedores comunicarse entre sí y entre otros contenedores, es posible publicar los puertos de los contenedores a nivel de la network. Esto facilita que tus contenedores puedan ser utilizados por otras redes.

Para crear networks con una gran cantidad de contenedores es recomendable utilizar el tipo de network llamado overlay.

```
$ docker network create --driver overlay webvinar-overlay-net
```

docker network connect/disconnect

Podemos conectar los contenedores una vez creados o cuando los desplegamos con `--net=NETWORK_NAME` .

```
$ docker network connect/disconnect NETWORK_NAME CONTAINER_NAME
```

Vemos el contenedor asignado a la red:

```
$ docker network inspect NETWORK_NAME $ docker inspect --format='{{json .NetworkSettings}}' CONTAINER_NAME | jq '.'
```

Ahora CONTAINER_NAME tendrá dos interfaces de red, una para la red creada y otra para la por defecto.

docker network rm

Cuando todos los contenedores de un red están parados o desconectados de la misma. Entonces, podremos borrar nuestra network usando el siguiente comando:

```
$ docker network rm NETWORK_NAME
```