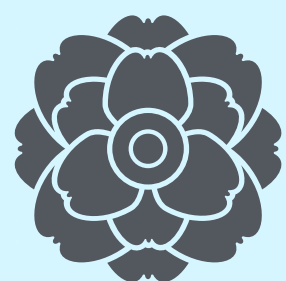


# Programowanie Aplikacji w Chmurze Obliczeniowej

## Laboratorium nr 10

Sterowniki sieciowe w środowisku docker. Tworzenie i wykorzystanie sieci mostkowych definiowanych przez użytkownika. Wolumeny i ich podział. Współdzielenie wolumenów. Integracja systemów plików hosta i kontenera - wolumeny typu bind mount

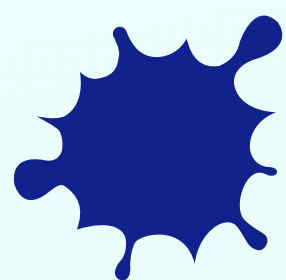
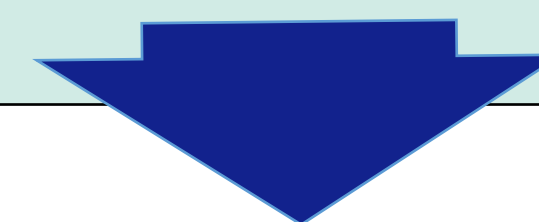
Dr inż. Sławomir Przyłucki  
[s.przylucki@pollub.pl](mailto:s.przylucki@pollub.pl)



## Docker - sterowniki sieciowe - cz. I

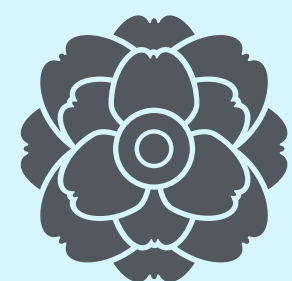
Sterownik	Opis
<b>Host</b>	Kontener wykorzystuje stos protokołów sieciowych hosta. Brak jest separacji przestrzeni nazw sieciowych (ang. network namespaces) co w praktyce oznacza, że wszystkie interfejsy sieciowe hosta mogą być bezpośrednio używane w kontenerze.
<b>Bridge</b>	W oparciu o ten sterownik, na hoście tworzony jest most (ang. bridge), którym zarządza daemon Docker. Domyślna konfiguracja tego mostu pozwala na wzajemną komunikację pomiędzy wszystkimi kontenerami, które są do niego przyłączone. Równolegle jest ustawione zezwolenie na komunikację do sieci zewnętrznych z wykorzystaniem mechanizmów NAT (ang. Network Address Translation).

Sterowni domyślny



Wszystkie kontenery uruchomiane bez jawnego przypisania do określonej sieci wykorzystują sterownik bridge i są przyłączone do domyślnego mostu (ang. bridge) o nazwie **docker0**





## Docker - sterowniki sieciowe - cz. II

### Overlay

Wykorzystując ten sterownik, tworzona jest sieć nakładkowa (ang. overlay network), która realizuje połączenia logiczne pomiędzy kontenerami. Struktura tej sieci jest kombinacją lokalnych mostów oraz sieci VXLAN w celu realizacji infrastruktury połączeń kontener – kontener odizolowanej od fizycznej infrastruktury wykorzystywanych połączeń sieciowych.

**Sterownik inicjalizowany domyślnie w przypadku wykorzystywania klastrów, np. klastra Swarm**

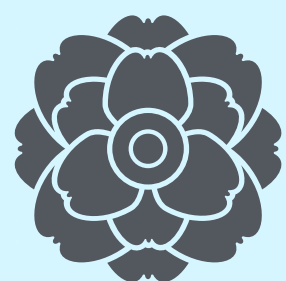
### MacVLAN

W tym trybie wykorzystywany jest dedykowany most MacVLAN, który łączy interfejs kontenera z interfejsem hosta. Stosowany do segmentacji sieci oraz zarządzaniem izolowanymi pulami adresowymi. Wspiera mechanizmy separacji ruchu zdefiniowane dla klasycznych sieci VLAN. IPVLAN dają z kolei pełną kontrolę na adresacją (IPv4 oraz IPv6) oraz konfiguracją routingu

**Sterownik bardzo przydatny (i popularny) w dedykowanych systemach wykorzystujących kontenery (np. rozwiązania z rodziny IoT)**

<https://docs.docker.com/network/drivers/macvlan/>

<https://docs.docker.com/network/drivers/ipvlan/>



## Docker - sterowniki sieciowe - cz. III

<b>None</b>	Sterownik ten tworzy sieciową przestrzeń nazw i pełny stos obsługi sieci wewnątrz kontenera. Natomiast nie jest tworzony interfejs sieciowy co sprawia, że kontener (bez dodatkowej konfiguracji) jest całkowicie izolowany od otaczającej go infrastruktury połączeń sieciowych.
-------------	---

**Sterownik wykorzystywany przy specyficznych (niestandardowych) rozwiązaniach systemowych w środowisku Docker**

<https://docs.docker.com/network/drivers/none/>

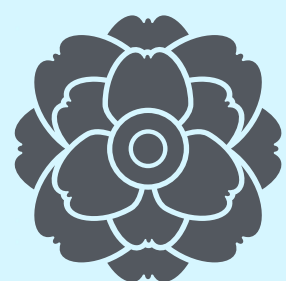
Referencyjna dokumentacja implementacji modelu CNM:

<https://github.com/moby/libnetwork/blob/master/docs/design.md>

Referencyjna dokumentacja wykorzystania poszczególnych sterowników sieciowych w środowisku Docker:

<https://docs.docker.com/network/network-tutorial-standalone/>

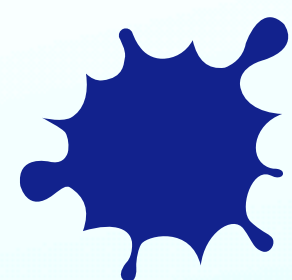




## Połączenia sieciowe - ustawienia domyślne

```
> ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: enp0s5: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 00:1c:42:49:55:43 brd ff:ff:ff:ff:ff:ff
    inet 10.211.55.13/24 metric 100 brd 10.211.55.255 scope global dynamic enp0s5
        valid_lft 1710sec preferred_lft 1710sec
    inet6 fdb2:2c26:f4e4:0:21c:42ff:fe49:5543/64 scope global dynamic mngtmpaddr noprefixroute
        valid_lft 2591916sec preferred_lft 604716sec
    inet6 fe80::21c:42ff:fe49:5543/64 scope link
        valid_lft forever preferred_lft forever
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default
    link/ether 02:42:bb:34:1d:ab brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
        valid_lft forever preferred_lft forever
```

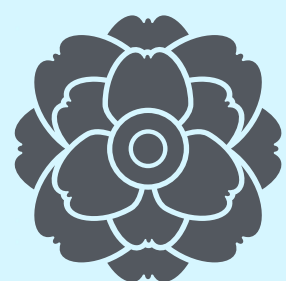
- W przypadku sterowników o zasięgu (ang. scope) „local”, wartość „network ID” jest unikalna dla każdego hosta.
- Dla zasięgu „swarm”, wartość „network ID” jest taka sama dla wszystkich maszyn w ramach danego klastra.



Po instalacji środowiska Docker (Docker Engine) w systemie operacyjnym tworzone jest most (ang. Bridge) o domyślnej nazwie docker0 (nie dotyczy instalacji opartych o Docker Desktop)

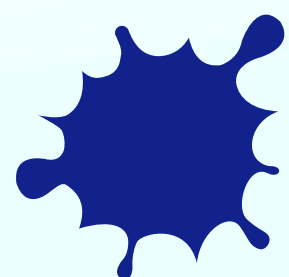
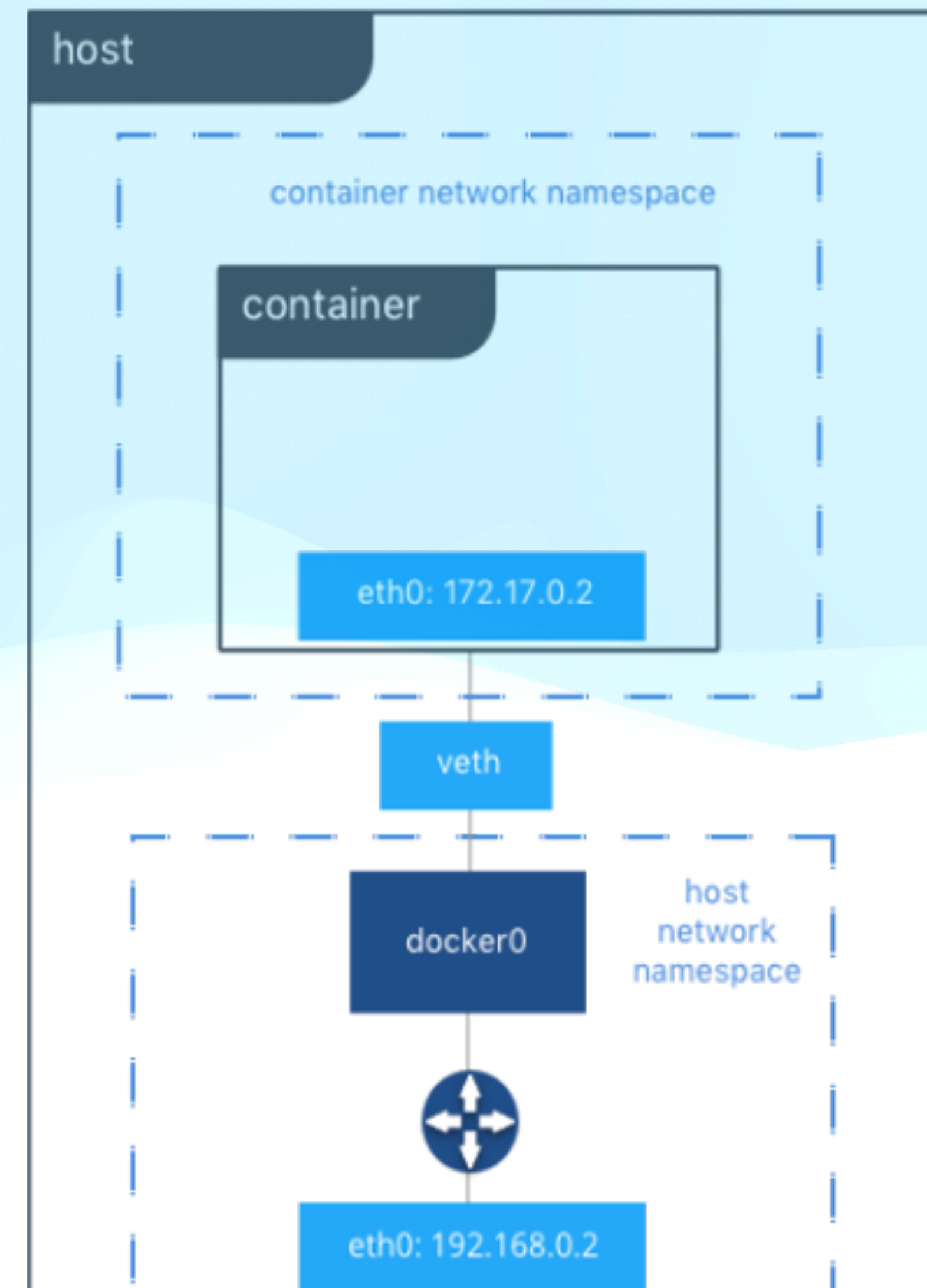
```
> docker network ls
NETWORK ID          NAME       DRIVER  SCOPE
b1d9ffc1d869        bridge    bridge  local
8a2ac4eb21e8        host      host    local
ce9bc3a85b4e        none     null    local
```





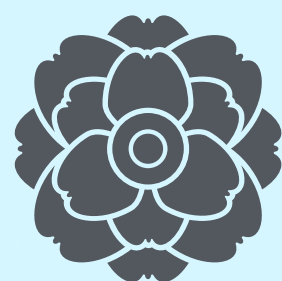
## Network Bridge Driver - cz. I

```
~  
> docker run --rm -it --name default alpine sh  
/ # ifconfig  
eth0      Link encap:Ethernet  HWaddr 02:42:AC:11:00:04  
          inet addr:172.17.0.4  Bcast:172.17.255.255  Mask:255.255.0.0  
          UP BROADCAST RUNNING MULTICAST  MTU:65535  Metric:1  
          RX packets:6 errors:0 dropped:0 overruns:0 frame:0  
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0  
          collisions:0 txqueuelen:0  
          RX bytes:516 (516.0 B)  TX bytes:0 (0.0 B)  
  
lo        Link encap:Local Loopback  
          inet addr:127.0.0.1  Mask:255.0.0.0  
          inet6 addr: ::1/128 Scope:Host  
          UP LOOPBACK RUNNING  MTU:65536  Metric:1  
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0  
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0  
          collisions:0 txqueuelen:1000  
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)  
  
/ # ip route  
default via 172.17.0.1 dev eth0  
172.17.0.0/16 dev eth0 scope link  src 172.17.0.4  
/ # █
```



Zakres adresów wykorzystywanych przez kontenery uruchomione w tym trybie są definiowane przez wewnętrzne sterowniki IPAM. Domyślnie, dla default bridge przypisana jest pula adresów: **172.[17-31].0.0/16** oraz **192.168.[0-240].0/20**





## Network Bridge Driver - cz. II

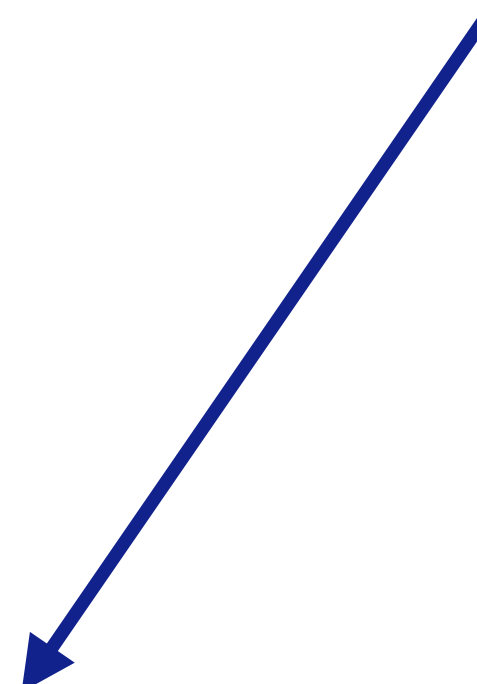
```
~  
> docker run -d --name new_default alpine sleep 3600  
2fbc44eaa2d19947aba37538bd0bb5b5978793e4b471b96a193985c1c74fba4
```

```
~  
> docker network inspect bridge
```

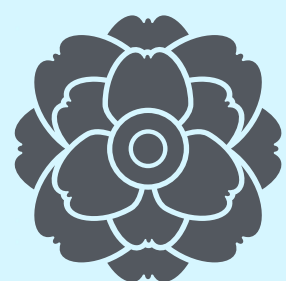
### CZĘŚĆ I

```
[  
  {  
    "Name": "bridge",  
    "Id": "0c37807346bbd5a085ae6891ba079bae2c3c8020d89ccff297a1e8f5078794fa",  
    "Created": "2024-05-19T21:44:32.979271625Z",  
    "Scope": "local",  
    "Driver": "bridge",  
    "EnableIPv6": false,  
    "IPAM": {  
      "Driver": "default",  
      "Options": null,  
      "Config": [  
        {  
          "Subnet": "172.17.0.0/16",  
          "Gateway": "172.17.0.1"  
        }  
      ]  
    },  
    "Internal": false,  
    "Attachable": false,  
    "Ingress": false,  
    "ConfigFrom": {  
      "Network": ""  
    },  
    "ConfigOnly": false,  
    "Containers": {  
      "2fbc44eaa2d19947aba37538bd0bb5b5978793e4b471b96a193985c1c74fba4": {  
        "Name": "new_default",  
        "EndpointID": "26175d031a187678307dd304e8f8fa54ae06db632aa38f5f00f86e3f3ba86775",  
        "MacAddress": "02:42:ac:11:00:04",  
        "IPv4Address": "172.17.0.4/16",  
        "IPv6Address": ""  
      }  
    }  
  }  
]
```

Lista i parametry kontenerów  
podłączonych do danej sieci







## Network Bridge Driver - cz. III

### CZĘŚĆ II

```
"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": "65535"
},
```

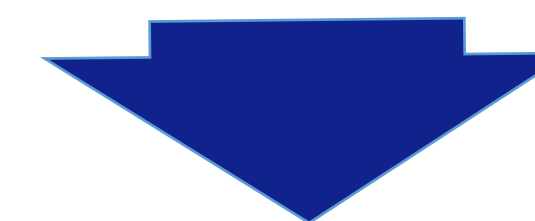
```
> docker network create --help
Usage: docker network create [OPTIONS] NETWORK
```

Create a network

Options:

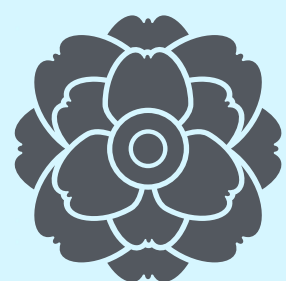
--attachable	Enable manual container attachment
--aux-address map	Auxiliary IPv4 or IPv6 addresses used by Network driver (default map[])
--config-from string	The network from which to copy the configuration
--config-only	Create a configuration only network
-d, --driver string	Driver to manage the Network (default "bridge")
--gateway strings	IPv4 or IPv6 Gateway for the master subnet
--ingress	Create swarm routing-mesh network
--internal	Restrict external access to the network
--ip-range strings	Allocate container ip from a sub-range
--ipam-driver string	IP Address Management Driver (default "default")
--ipam-opt map	Set IPAM driver specific options (default map[])
--ipv6	Enable IPv6 networking
--label list	Set metadata on a network
-o, --opt map	Set driver specific options (default map[])
--scope string	Control the network's scope
--subnet strings	Subnet in CIDR format that represents a network segment

Zmiana parametrów (opcji) danej sieci jest możliwa podczas tworzenia własnej sieci



[https://docs.docker.com/engine/reference/commandline/network\\_create/#bridge-driver-options](https://docs.docker.com/engine/reference/commandline/network_create/#bridge-driver-options)





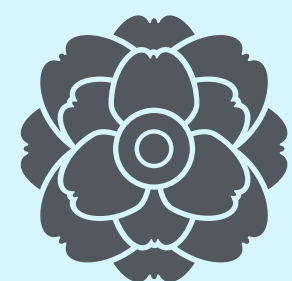
## Konfiguracja sieci - przykład - zmiana MTU

Częstym problemem podczas obsługi kontenerów w danej infrastrukturze sieciowej jest ustawienie własnej wartości MTU (np. równej 9000). Demon Dockera nie sprawdza MTU połączenia wychodzącego. Dlatego wartość MTU jest ustawiona na domyśle 1500.

```
~
> docker network create -o "com.docker.network.driver mtu=9000 skynet
45686298a893dd8e47eae8aa98f3623e7a7a34fcad782ff88078ea45e6df32f2
~
> docker network ls
NETWORK ID          NAME       DRIVER      SCOPE
0c37807346bb        bridge    bridge      local
cd47a3e5c77f        host      host        local
1f1079fb7a88        minikube  bridge      local
d2ac7b0a36cf        none     null        local
45686298a893        skynet    bridge      local
~
> docker network inspect skynet | jq '.[0].Options'
{
  "com.docker.network.driver mtu": "9000"
}
```

Przykład: utworzenie własnej sieci wykorzystującej sterownik bridge z MTU=9000 (tzw. Jumbo frames)

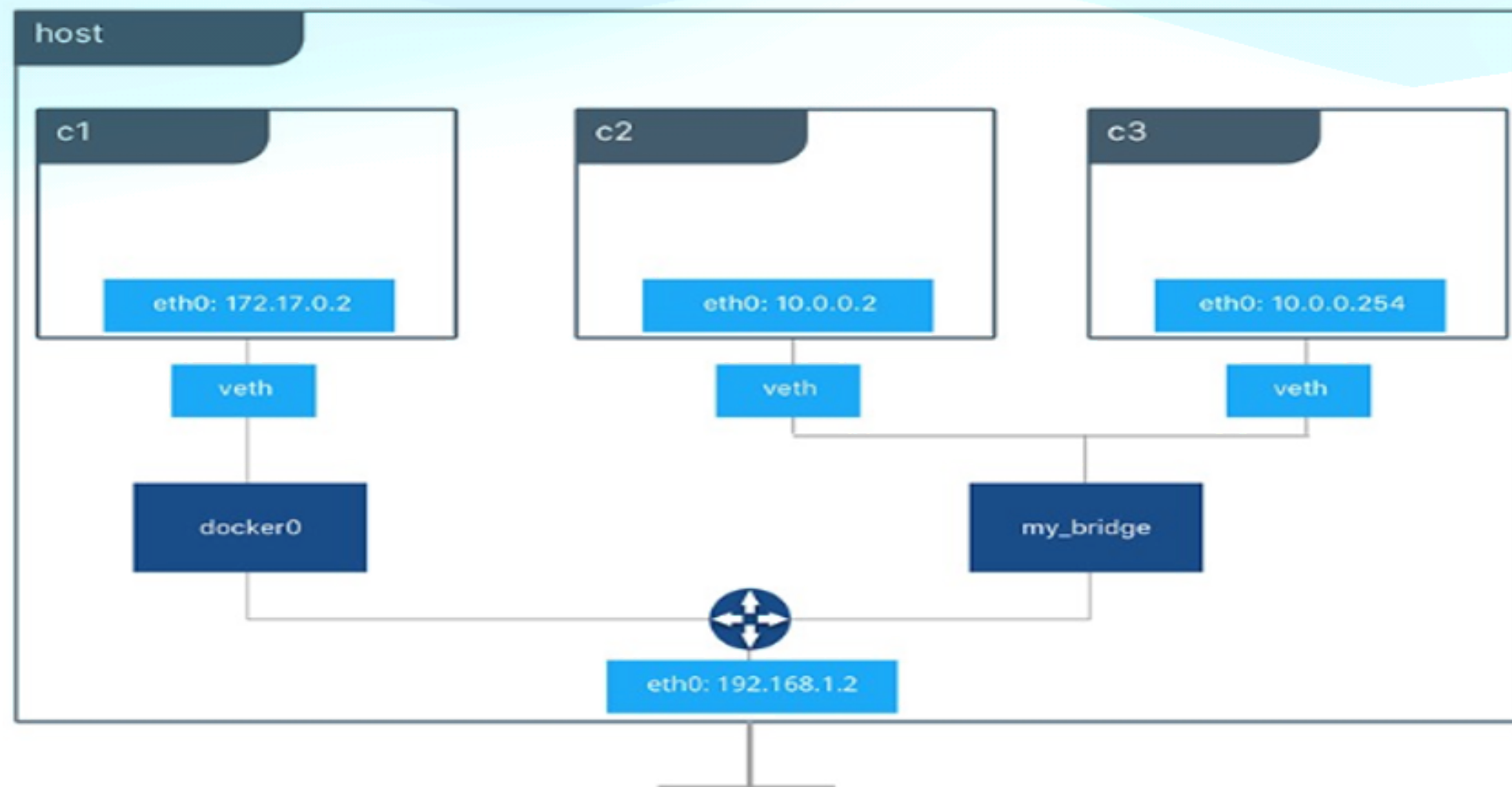




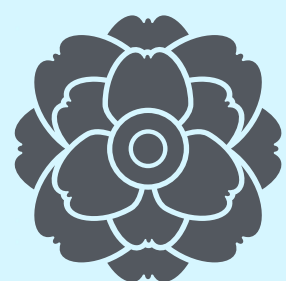
## Sieci mostkowe, definiowane przez użytkownika - cz. I

Środowisko Docker daje możliwość definiowania własnych sieci dowolnego typu. Jednym z nich może być tryb mostu definiowanego przez użytkownika. Pośród jego wielu zalet, istotna jest możliwość ręcznego przypisywania adresów (pojedynczych adresów jak i puli adresów) oraz odkrywanie usług po nazwach.

<https://docs.docker.com/network/drivers/bridge/#differences-between-user-defined-bridges-and-the-default-bridge>







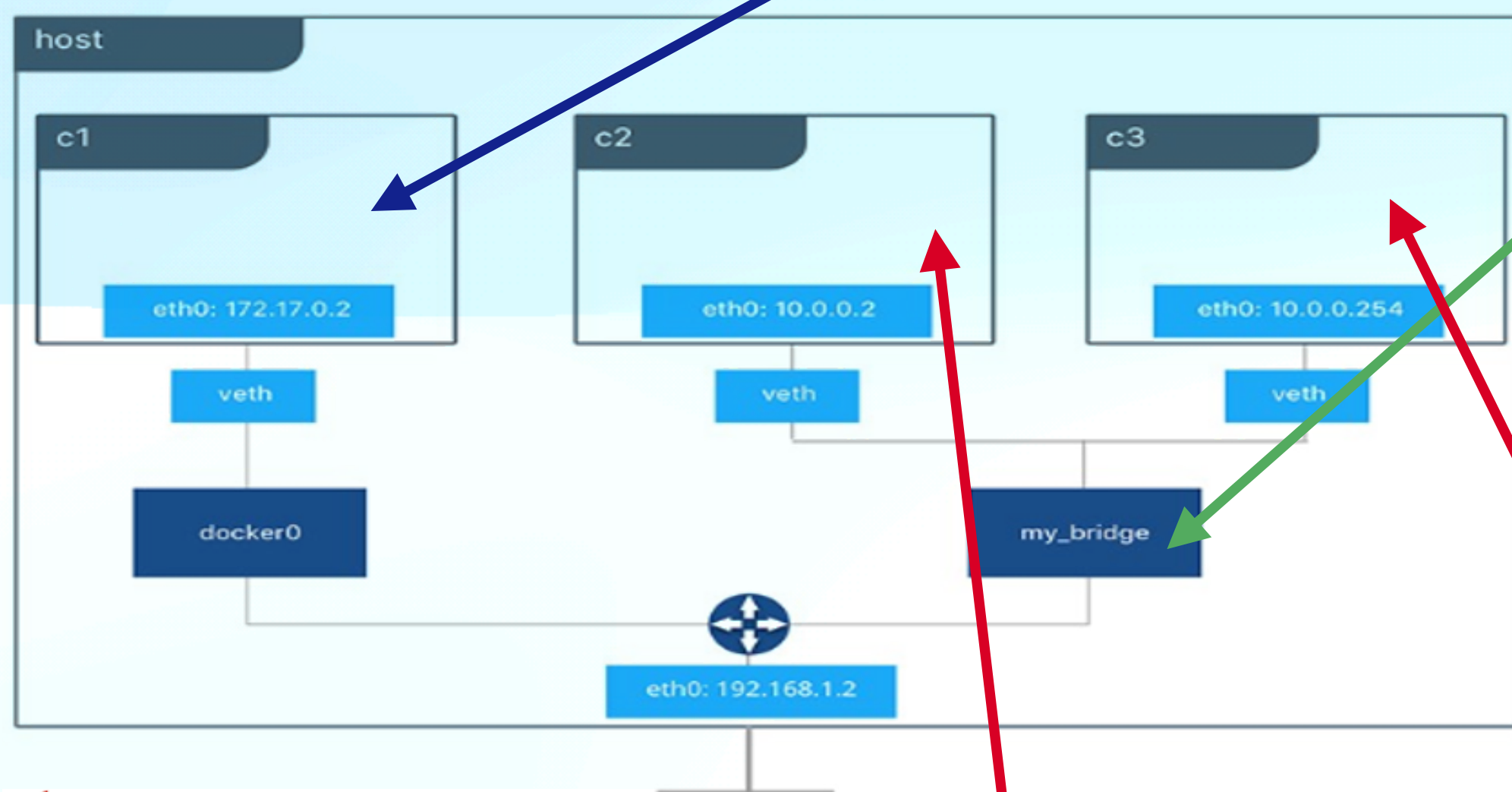
## Sieci mostkowe, definiowane przez użytkownika - cz. II

1

```
> docker network create --driver=bridge --subnet=10.0.0.0/24 my_bridge  
136a4110981f38a3334f021948f56372c43998685e3d52425d3b717b01695420
```

2

```
> docker run --rm -dt --name c1 alpine sh  
c2469d2677c33de55224747456e3d963b3fdc8f108cab15d78c5e840c7b02b08
```



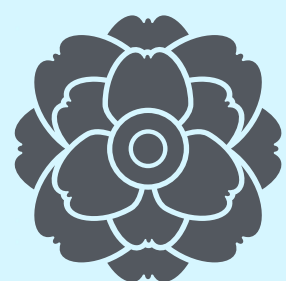
3

```
> docker run --rm -dt --name c2 --network=my_bridge --ip=10.0.0.2 alpine sh  
bcf4d9dcd604f8e07dbe0ad17ee93bd40ce5ad54047fb157d115c7dd4de0728f
```

4

```
> docker run --rm -dt --name c3 --network=my_bridge --ip=10.0.0.254 alpine sh  
bf2f8c304d867d3f109b2abc3c20510641b29e5668ea25910ad551d73a01ef06
```





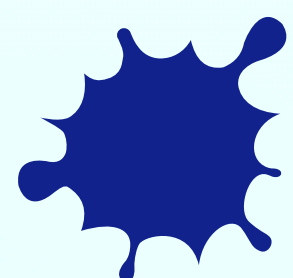
## Sieci mostkowe, definiowane przez użytkownika - cz. III

5

```
~  
> docker network inspect my_bridge | jq '.[].Containers'  
{  
  "bcf4d9dcd604f8e07dbe0ad17ee93bd40ce5ad54047fb157d115c7dd4de0728f": {  
    "Name": "c2",  
    "EndpointID": "da23f76ac41968ed9844ba0346e2265b8f4f9946e93ee05a7ffe888bd80d05d9",  
    "MacAddress": "02:42:0a:00:00:02",  
    "IPv4Address": "10.0.0.2/24",  
    "IPv6Address": ""  
  },  
  "bf2f8c304d867d3f109b2abc3c20510641b29e5668ea25910ad551d73a01ef06": {  
    "Name": "c3",  
    "EndpointID": "cdb27005b0a35bac9f7c30b7cdc9d42a7d5ca5354116ed0f87c402f3f66ba893",  
    "MacAddress": "02:42:0a:00:00:fe",  
    "IPv4Address": "10.0.0.254/24",  
    "IPv6Address": ""  
  }  
}
```

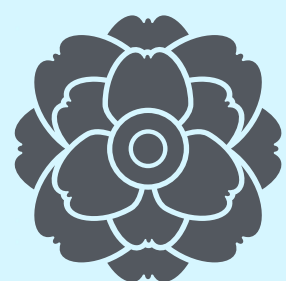
```
docker network connect <NETWORK> <CONTAINER>  
docker network disconnect <NETWORK> <CONTAINER>
```

```
~  
> docker network ls  
NETWORK ID        NAME        DRIVER    SCOPE  
0c37807346bb      bridge     bridge    local  
cd47a3e5c77f      host       host      local  
1f1079fb7a88      minikube   bridge    local  
136a4110981f      my_bridge  bridge    local  
d2ac7b0a36cf      none      null      local
```



Tryb tryb mostu definiowanego przez użytkownika umożliwia przyłączanie do sieci i odłączenia kontenera od sieci w trakcie jego działania. W pozostałych trybach (np. w trybie mostu domyślnego) należy zatrzymać kontener i przyłączyć go do innej sieci.





## Tworzenie wolumenów - cz. I

```
~
> docker volume create testV1
testV1
~
> docker volume ls
DRIVER      VOLUME NAME
local       7933cc3c3323223fb4725b44a8ffa8fb2de6eac45c872d850c13672336ce6cdc3
local       buildx_buildkit_labbuilder0_state
local       testV1
~
> docker volume inspect testV1
[
  {
    "CreatedAt": "2023-04-23T19:50:08Z",
    "Driver": "local",
    "Labels": null,
    "Mountpoint": "/var/lib/docker/volumes/testV1/_data",
    "Name": "testV1",
    "Options": null,
    "Scope": "local"
  }
]
```

```
~
> docker volume --help

Usage:  docker volume COMMAND

Manage volumes

Commands:
  create      Create a volume
  inspect     Display detailed information on one or more volumes
  ls          List volumes
  prune       Remove all unused local volumes
  rm          Remove one or more volumes
```

**Deklaracja innego sterownika wolumenu (ang. volume driver) jest możliwa tylko i wyłącznie w trakcie tworzenia wolumenu jako samodzielnego zadania (czyli w poleceniu `docker volume create`)**

**Dodatkowe opcje dla sterowników innych niż local**

<https://docs.docker.com/storage/volumes/#use-a-volume-driver>

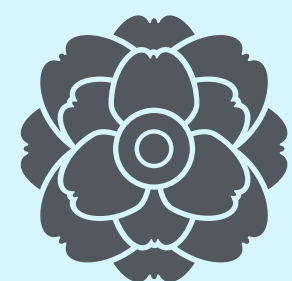
```
~
> docker volume create --help

Usage:  docker volume create [OPTIONS] [VOLUME]

Create a volume

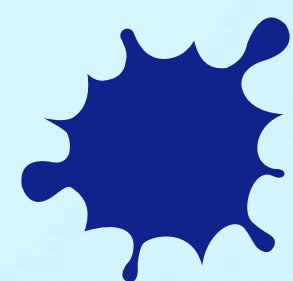
Options:
  -d, --driver string  Specify volume driver name (default "local")
  --label list         Set metadata for a volume
  -o, --opt map        Set driver specific options (default map[])
```





## Przyłączanie kontenera do utworzonego wolumenu - cz. I

Aby wykorzystać utworzony wolumen należy stworzyć nowy kontener, np. na bazie obrazu Ubuntu, i zamontować utworzony wcześniej wolumen do tego kontenera. W tym celu wykorzystywana jest opcja `-v` lub opcja `--mount` w poleceniu `docker run`

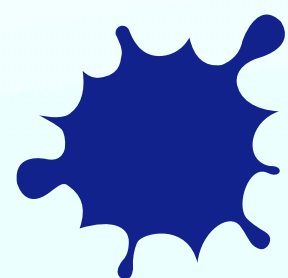


Proszę zapoznać się z fragmentem dokumentacji (link poniżej). Obecnie obie opcje są poprawne ale zalecane jest używanie opcji `--mount` ponieważ tylko ona jest wykorzystywana w przypadku korzystania z wolumenów w architekturach klastrowych (np. Docker Swarm).

<https://docs.docker.com/storage/volumes/#choose-the--v-or---mount-flag>

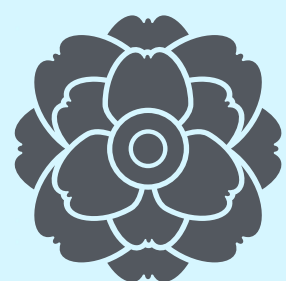
Składnia opcji `--mount` wymaga podania pary kluczy, `source` oraz `target`:

`source=<nazwa wolumenu>,target=<miejsce mountowania w kontenerze>`



Ścieżka w kluczu `target` powinna być ścieżką bezwzględną (zaczynać się od `/`). Jeżeli podana ścieżka nie istnieje to zostanie ona dodana w wyniku działania polecenia `docker run`





## Przyłączanie kontenera do utworzonego wolumenu - cz. II

```
~  
> docker run -it --name voltest \  
--mount source=testV1,target=/magazyn \  
alpine  
/ # ls -al | grep magazyn  
drwxr-xr-x    2 root    root          4096 .magazyn  
/ #
```

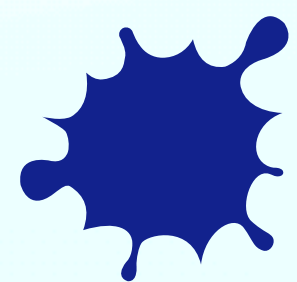
Wolumen utworzy katalog w ścieżce  
zdefiniowanej w kluczu `target`

Wolumen jest zarządzany przez daemona  
Docker i przechowywany w domyślnej  
ścieżce w systemie plików hosta

Ustawienia domyślne dla wolumenu to:

- tryb (ang. Mode): `z` —> oznacza zezwolenie  
na współdzielenie wolumenu
- uprawnienia: `RW`

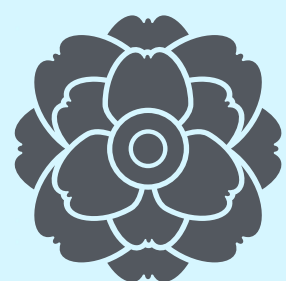
```
~  
> docker container inspect voltest | jq '[][.Mounts]'  
[  
  {  
    "Type": "volume",  
    "Name": "testV1",  
    "Source": "/var/lib/docker/volumes/testV1/_data",  
    "Destination": "/magazyn",  
    "Driver": "local",  
    "Mode": "z",  
    "RW": true,  
    "Propagation": ""  
  }  
]
```



Warto zapoznać się z dodatkowymi  
przykładami:

<https://github.com/rhatdan/moby1/blob/e6473011583967df4aa5a62f173fb421cae2bb1e/docs/sources/reference/commandline/cli.md#run>





# PAwChO – Laboratorium 10

## Współdzielenie wolumenów

```
~  
> docker run -dt --rm --name voltestA --mount source=testV1,target=/magazynA ubuntu /bin/bash  
7b51632d8289435093aeab9b4f492d209ecc4e9c3d8959330c3a0bce33cae159
```

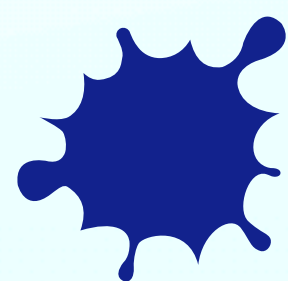
```
~  
> docker run -it --rm --name voltestB --volumes-from voltestA ubuntu /bin/bash  
root@f01e4e25524e:/# ls -al | grep magazynA  
drwxr-xr-x  2 root root 4096 Apr 23 19:50 magazynA  
root@f01e4e25524e:/#
```

<https://docs.docker.com/storage/volumes/#share-data-between-machines>

```
~  
> docker container inspect voltestA | jq '.[].Mounts'  
[  
  {  
    "Type": "volume",  
    "Name": "testV1",  
    "Source": "/var/lib/docker/volumes/testV1/_data",  
    "Destination": "/magazynA",  
    "Driver": "local",  
    "Mode": "z",  
    "RW": true,  
    "Propagation": ""  
  }  
]
```

```
~  
> docker container inspect voltestB | jq '.[].Mounts'  
[  
  {  
    "Type": "volume",  
    "Name": "testV1",  
    "Source": "/var/lib/docker/volumes/testV1/_data",  
    "Destination": "/magazynA",  
    "Driver": "local",  
    "Mode": "",  
    "RW": true,  
    "Propagation": ""  
  }  
]
```

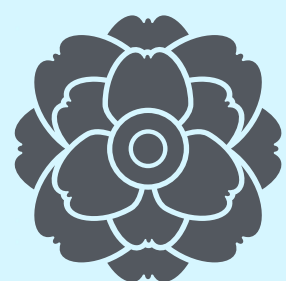
UWAGA: brak flagi „z”



W przypadku współdzielenia wolumenów, bezpieczna konfiguracja to prawa dostępu „read-only”. Sposób deklarowania takiego atrybutu wolumenu:

<https://docs.docker.com/storage/volumes/#use-a-read-only-volume>

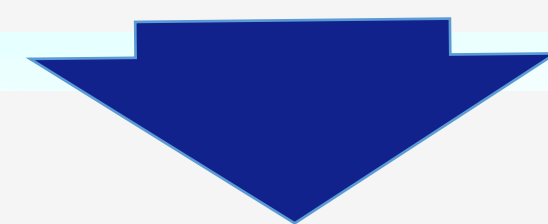




## Tworzenie wolumenu wraz z kontenerem

Istnieje możliwość tworzenia wolumenów w trakcie tworzenie samego kontenera Docker. Ponownie wykorzystywana jest opcja `-v` lub `--mount` w poleceniu `docker run` a różnica polega na tym, że w kluczu `source` podawana jest nazwa wolumenu, który nie został wcześniej utworzony.

```
➤ docker run -it --rm --name voltestC --mount source=newVol,target=/newdir ubuntu /bin/bash
root@258574e26499:/# ls -al | grep newdir
drwxr-xr-x  2 root root 4096 Apr 23 21:01 newdir
root@258574e26499:/#
```



```
➤ docker volume inspect newVol
[
  {
    "CreatedAt": "2023-04-23T21:01:35Z",
    "Driver": "local",
    "Labels": null,
    "Mountpoint": "/var/lib/docker/volumes/newVol/_data",
    "Name": "newVol",
    "Options": null,
    "Scope": "local"
  }
]
```



Nie można usunąć wolumenu, który jest wykorzystywany. Należy wcześniej usunąć kontener, który używa ten wolumen.

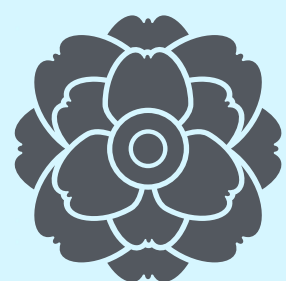
**Usunięcie wolumenu:**

`docker volume rm <nazwa wolumenu>`

**Usunięcie wszystkich  
niewykorzystywanych wolumenu:**

`docker volume prune`





# PAwChO — Laboratorium 10

## Wolumeny typu bind mount

```
🍏 ~/Labs/lab10
> mkdir -p ~/Labs/lab10/nglogs
🍏 ~/Labs/lab10
> docker run -d --name ngx \
--mount type=bind,source=/Users/slawek/Labs/lab10/nglogs,target=/var/log/nginx \
-p 4002:80 nginx
b5f3cdf7e44e273aeb2a5d1c131efba94347d0d72224009c5c124e6255b7d8ca
```

↓

```
🍏 ~/Labs/lab10
> ls ~/Labs/lab10/nglogs
access.log error.log
🍏 ~/Labs/lab10
> docker exec ngx ls /var/log/nginx
access.log
error.log
```

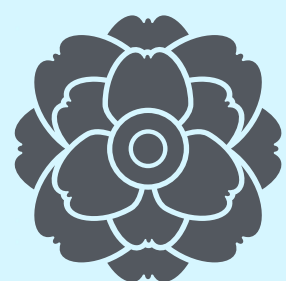
→

```
🍏 ~/Labs/lab10
> docker container inspect ngx | jq '[][.Mounts]'
[
  {
    "Type": "bind",
    "Source": "/Users/slawek/Labs/lab10/nglogs",
    "Destination": "/var/log/nginx",
    "Mode": "",
    "RW": true,
    "Propagation": "rprivate"
  }
]
```



Korzystanie z wolumenów typu bind mount wymaga by był wcześniej utworzony katalog w systemie macierzystym oraz by miał on prawa dostępu zgodne z zaplanowanym wykorzystaniem tego wolumenu






# Deklaracje wolumenów w plikach Dockerfile

Przykład: Należy odnaleźć na DockerHub obraz oficjalny bazy Redis a następnie w zakładce *arch* wybrać np. amd64.

<https://hub.docker.com/layers/library/redis/latest/images/sha256-9341b6548cc35b64a6de0085555264336e2f570e17ecff20190bf62222f2bd64?context=explore>



redis:latest

MULTI-PLATFORM

INDEX DIGEST sha256:5a93f6b2e391b78e8bd3f9e7e1e1e06aeb5295043b4703fb88392835cec924a0

OS/ARCH

linux/amd64

COMPRESSED SIZE

43.4 MB

LAST PUSHED

5 days ago by [doijanky](#)

TYPE

Image

VULNERABILITIES

2

29

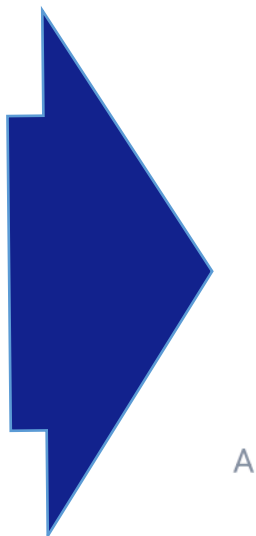
11

26

6

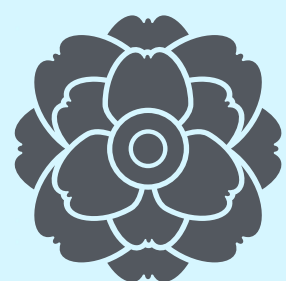
MANIFEST DIGEST

sha256:9341b654...



Layers (17)				
↳	3	RUN /bin/sh -c set -eux; apt-get update; apt-get insta...	872 B	✓
↳	4	ENV GOSU_VERSION=1.17	0 B	✓
↳	5	RUN /bin/sh -c set -eux; savedAptMark="\$(apt-mark ...	1.44 MB	!
↳	6	ENV REDIS_VERSION=7.2.4	0 B	✓
↳	7	ENV REDIS_DOWNLOAD_URL=http://download.redis....	0 B	✓
↳	8	ENV REDIS_DOWNLOAD_SHA=8d104c26a154b29fd...	0 B	✓
↳	9	RUN /bin/sh -c set -eux; savedAptMark="\$(apt-mark ...	14.92 MB	!
↳	10	RUN /bin/sh -c mkdir /data && chown redis:redis /da...	97 B	✓
↳	11	VOLUME [/data]	0 B	✓
↳	12	WORKDIR /data	32 B	✓
↳	13	COPY docker-entrypoint.sh /usr/local/bin/ # buildkit	570 B	✓
↳	14	ENTRYPOINT ["docker-entrypoint.sh"]	0 B	✓
↳	15	EXPOSE map[6379/tcp:{}]	0 B	✓
↳	16	CMD ["redis-server"]	0 B	✓





## Zadanie OBOWIĄZKOWE

Należy uruchomić trzy kontenery o nazwach odpowiednio: **web1**, **web2**, **web3**, które zawierać będą serwery `nginx` w wersji *latest* w taki sposób by:

- wszystkie te serwery były połączone do jednej sieci mostkowej definiowanej przez użytkownika (nazwa sieci: **lab10net**),
- wszystkie serwery były dostępne z sieci zewnętrznej (z poziomu używanego komputera, laptopa),
- poszczególne serwery wyświetlały prosta stronę html zawierającą: *numer laboratorium, imię i nazwisko studenta* a strona ta została połączona do każdego z serwerów `nginx` wykorzystując wolumeny z uprawnieniami dostępu: **read-only**,
- poszczególne serwery zapisywały logi do trzech dedykowanych katalogów w podkatalogu katalogu domowego o nazwie **lab10**. Katalog **lab10** ma zostać dołączony do kontenerów również wykorzystując wolumeny.



W sprawozdaniu proszę umieścić wszystkie użyte polecenia wraz z wynikiem ich działania. Polecenia te powinny dowieść, że wszystkie trzy serwery poprawnie wyświetlają stronę html a logi serwerów zostały poprawnie zapisane i są dostępne w systemie macierzystym (z poziomu wykorzystywanego komputera, laptopa)