LABORATORIUM PROGRAMOWANIA W CHMURACH OBLICZENIOWYCH

LABORATORIUM NR 3.

Konfiguracja połączeń sieciowych i komunikacji w środowisku Docker.

Wymiana danych pomiędzy kontenerami jak otoczeniem może być zorganizowana w oparciu o dwa podstawowe rozwiązania:
- tryby sieciowe pracy interfejsów wirtualnych kontenerów,
- mechanizm łączy.

UWAGA: Druga z wymienionych metodjest metodą nie zalecaną (ang. legacy solution) i nie jest dalej rozwijana. Z tego względu, w niniejszym sprawozdaniu będzie tylko wspomniana z kilkoma podstawowymi przykładami)

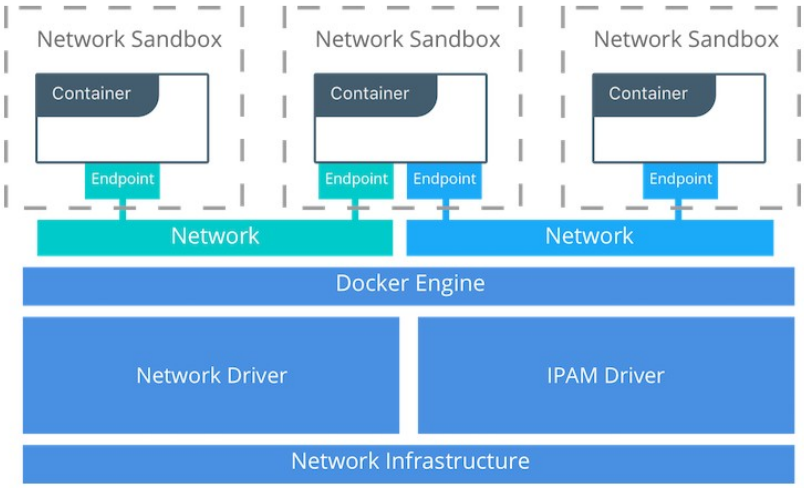
W chwili obecnej, w odniesieniu do kontenerów Docker (ale także innych, konkurencyjnych rozwiązań kontenerów programowych) wykorzystywane są dwa modele organizacji współpracy z sieciami. Są to odpowiednio:

- Container Networking Model (CNM) - https://github.com/docker/libnetwork/blob/master/docs/design.md

- Container Network Interface (CNI) - https://github.com/containernetworking/cni

Środowisko Docker CE, używane na laboratoriach, wykorzystuje pierwszy z wymienionych modeli. Model ten został zaimplementowany w bibliotece „libnetwork” będącej jednym z odstawowych elementów środowiska Docker (Docker engine).

Filozofia CNM polega na umożliwieniu jak największej uniwersalności implementacji połączeń sieciowych tak by aplikacje mogły współpracować (być dołączone) do dowolnej infrastruktury teleinformatycznej. Struktura modelu CNM jest przedstawiona na rysunku poniżej.



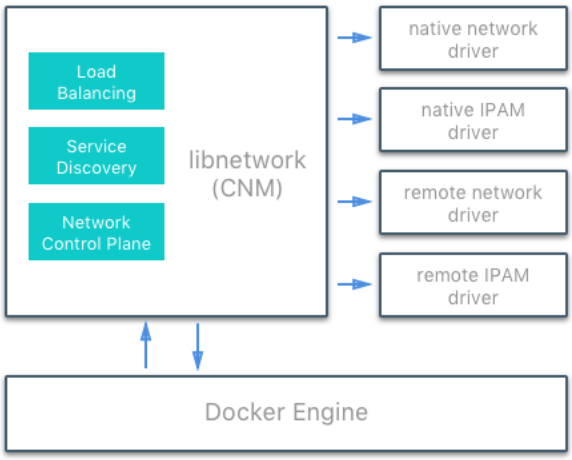
W skład modeli wchodzi kilka elementów. Istotna ich cechą jest to, że są one niepowiązane (nie opierają się) z szczegółowymi rozwiązaniami stosowanymi w dzisiejszych systemach operacyjnych i mechanizmach sieciowych (ang. agnostic components). Do najważniejszych elementów należą:

- **Sandbox** — Sandbox zawiera konfigurację stosu sieciowego kontenera. Obejmuje ona np. wirtualne interfejsy kontenera, tablice routingu czy też ustawienia infrastruktury DNS. W przypadku wykorzystywanego, macierzystego środowiska Docker opartego o Jądro Linux, implementacja Sandbox wykorzystuje Linux Network Namespace. Podstawową zaletą takiego podejścia jest to, że Sandbox może definiować wiele punktów końcowych (ang. endpoinyts), które z kolei mogą być przyłączone do wielu, różnych sieci.

- **Endpoint** — Endpoint jest łącznikiem pomiędzy Sandbox a dana siecią. W praktyce oznacza to, że jest abstrakcją, którą widzi aplikacja w kontenerze jako dostępne połączenie sieciowe, bez poznawania szczegółów funkcjonowania tego połączenia. Z oczywistych względów pozwala to na zachowanie przenośności kontenerów (i zawartych w nich aplikacji) pomiędzy różnymi infrastrukturami sieciowymi.

- **Network** — Model CNM okresla pojęcia sieci (Network) w rozumieniu modelu warstwowego OSI. W najkrótszym ujęciu jest to zbiór endpoints mający możliwość wzajemnego łączenia się i wymiany danych poprzez mechanizmy sieciowe. W praktyce, środowisko Docker wykorzystuje wiele rozwiązań realizacji takich sieci, np. od najprostszych jak Linux bridge czy też VLAN do bardziej zaawansowanych, wykorzystywanych w strukturach klastrowych.

Realizacja połączenia endpoints opiera się na wykorzystaniu sterowników sieciowych. Schematycznie ilustruje to rysunek poniżej:



Obecnie, w implementacja CNM w postaci biblioteki „libnetwork” definiuje następujący zestaw sterowników:

- **Network Drivers** — zestaw tzw. Docker Network Drivers, który dostarcza metod wymiany danych (przyłączenia się) do różnych sieci. Każdy sterownik wchodzący w skład tego zestawu oferuje konkretna funkcjonalność powiązana z danym typem infrastruktury sieciowej. Mogą być one wykorzystywane wymiennie (ang. pluggable solution) w zależności od specyfiki danego projektu. Istotne jest to, że jednocześnie wiele sterowników może być używane przez dany Docker Engine lub klaster.

UWAGA: Każda, pojedyncza sieć zdefiniowana w ramach środowiska Docker (lub klastra) musi być powiązana z tylko jednym sterownikiem.

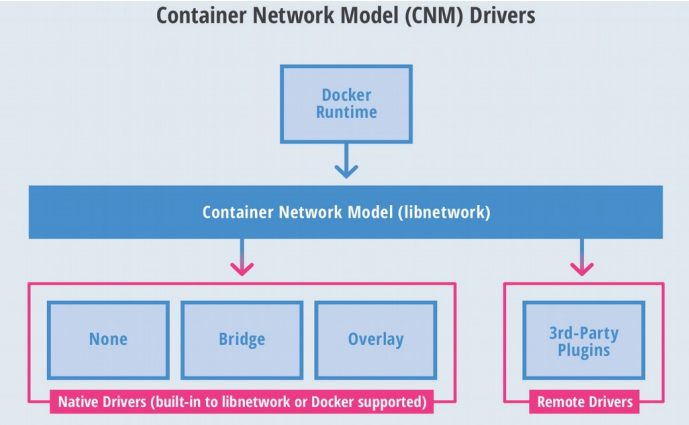
Wśród zestawy Nwtwork Drives, model CNM definiuj dwie podstawowe podgrupy:

- **Native Network Drivers** — jak wskazuje nazwa, Native Network Drivers są integralną częścią Docker Engine i są rozwijane przez Docker.

- **Remote Network Drivers** — Remote Network Drivers są sterownikami opracowanymi przez społeczność programistów lub firmy zajmujące się dostarczaniem i rozwijaniem dedykowanych rozwiązań sieciowych. Istnieje możliwość dołączenia do tej grupy własnych rozwiązań sterowników, dedykowanych dla określonej infrastruktury sieciowej.

IPAM Drivers — Docker posiada własne, wbudowane rozwiązania z grupy IPAM (ang. IP Address Management), które definiują domyślne zakresy adresów w danych podsieciach i adresy dla niezbędnych składników sieciowych (jeśli nie są one jednoznacznie określone w danym projekcie). Podobnie jak w przypadku sterowników sieciowych, model CNM dopuszcza stosowanie zewnętrznych rozwiązań (zestaw taki określany jest jako remote IPAM)

Powyższą strukturę przyjętego w CNM rozwiązania, ilustruje rysunek poniżej.



Poniżej jest przedstawiona pełna lista natywnych sterowników sieciowych, dostępnych w środowisku Docker wykorzystującym model CNM (dostępnych dla wersji Docker EE).

Driver	Description
Host	With the `host` driver, a container uses the networking stack of the host. There is no namespace separation, and all interfaces on the host can be used directly by the container
Bridge	The `bridge` driver creates a Linux bridge on the host that is managed by Docker. By default containers on a bridge can communicate with each other. External access to containers can also be configured through the `bridge` driver
Overlay	The `overlay` driver creates an overlay network that supports multi-host networks out of the box. It uses a combination of local Linux bridges and VXLAN to overlay container-to-container communications over physical network infrastructure
MACVLAN	The `macvlan` driver uses the MACVLAN bridge mode to establish a connection between container interfaces and a parent host interface (or sub-interfaces). It can be used to provide IP addresses to containers that are routable on the physical network. Additionally VLANs can be trunked to the `macvlan` driver to enforce Layer 2 container segmentation
None	The `none` driver gives a container its own networking stack and network namespace but does not configure interfaces inside the container. Without additional configuration, the container is completely isolated from the host networking stack

Listę sterowników obecnych w danym systemie można uzyskać za pomocą polecenia:

$ docker network ls

Dla systemu Docker CE

```
student@student-PwCh0:~$ docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
aaf166163dfd        bridge              bridge              local
be91c7e8ade7        host                host                local
3617fd744479        none                null                local
```

Dla systemu Docker EE

```
$ docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
1475f03fbecb        bridge              bridge              local
e2d8a4bd86cb        docker_gwbridge     bridge              local
407c477060e7        host                host                local
f4zr3zrswlyg        ingress             overlay             swarm
c97909a4b198        none                null                local
```

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

Listę dostępnych sterowników należących do grupy „Remote Network Drives” można przejrzeć np. na Docker Store, pod adresem:
https://store.docker.com/search?q=&type=plugin&category=network

A. Ttryby sieciowe
W tej części laboratorium zostaną przedstawione tryby sieciowe z ograniczeniem do tych sterowników, które mogą być wykorzystywane na pojedynczym hoście. Rozwiązania wielo-hostowe (w tym klastry Swarm) zostaną przedstawione na jednym z kolejnych ćwiczeń. Pominięte też są rozwiązania oparte o sterownik MACVLAN ponieważ deklaracje tego typu się nie są dostępne w większości implementacji chmurowych (ograniczenia narzucane przez operatorów chmur komputerowych). Zainteresowani mogą znaleźć informacje o wykorzystaniu tego sterownika pod adresem: https://docs.docker.com/v17.09/engine/userguide/networking/get-started-macvlan/

None (Isolated) Network Driver

Ten sterownik (ten tryb sieciowy) nie oferuje żadnych opcji konfiguracyjnych. Podczas jego stosowania Docker Engine nie tworzy żadnego interfejsu wirtualnego wewnątrz kontenera poza standartowym interfejsem petli wewnętrznej (loopback). Oznacza to, że kontener jest całkowicie izolowany od pozostałych kontenerów oraz od systemu (hosta) macierzystego. Deklaracja wykorzystania tego trybu sieciowego polega na dodaniu do polecenia uruchamiającego kontener opcji –net=none

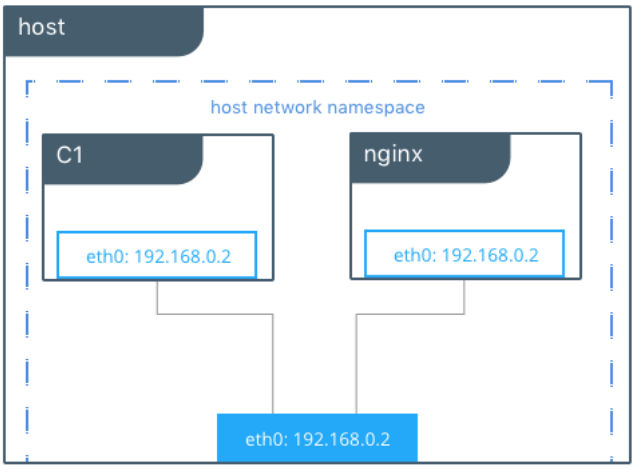
```
student@student-PwCh0:~$ docker run --rm -it --net=none --name=test_none busybox:latest sh
/ # ifconfig -a
lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          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)

/ # exit
student@student-PwCh0:~$
```

UWAGA1: W tym trybie, sterownik tworzy oddzielną przestrzeń nazw (namespace) dla kontenera co gwarantuje pełną, sieciową izolację kontenera.

UWAGA2: Użycie sterownika sieci „none” uniemożliwia podłączenie kontenera do jakiejkolwiek innej sieci.

Host Network Driver

Sterownik host jest (obok trybów sieciowych opartych na mostach) najpopularniejszym rozwiązaniem wśród developerów wykorzystujących środowisko Docker. Idea działania tego sterownika polega na całkowitym wyłączeniu mechanizmów obsługi połączeń sieciowych w kontenerze a sam kontener wykorzystuje stos obsługi sieci, skonfigurowany w systemie operacyjnym hosta macierzystego. W związku z tym nie tworzona jest przestrzeń nazwa (network namespace) dla kontenerów pracujących w tym trybie a wszystkie kontenery w tym trybie pracują w przestrzeni nazw systemu macierzystego. W tym trybie wszystkie kontenery „widzą się” za pośrednictwem interfejsy hosta macierzystego. Ilustruje to rysunek poniżej (adresy są przykładowe).



UWAGA: Ponieważ połączenia „przechodzą” przez interfejs hosta macierzystego to poszczególne kontenery nie mogą korzystać z tych samych portów.

Bazując na rysunku powyżej, przykładowe działanie tego trybu można zilustrować w następujący sposób.

```
student@student-PwCh0:~$ docker run -itd --net host --name c1 alpine sh
21edbb75e35dec9e201c9012cbb4b564e6badf4d74bc658dd7f9bb1f53d4904f
student@student-PwCh0:~$ docker run -itd --net host --name nginx nginx
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
f17d81b4b692: Pull complete
d5c237920c39: Pull complete
a381f92f36de: Pull complete
Digest: sha256:b73f527d86e3461fd652f62cf47e7b375196063bbbd503e853af5be16597cb2e
Status: Downloaded newer image for nginx:latest
1127ac47089c0aa28ccee780e5a6c9788ec710a705ff70a6707e63b7bd80a70f
```

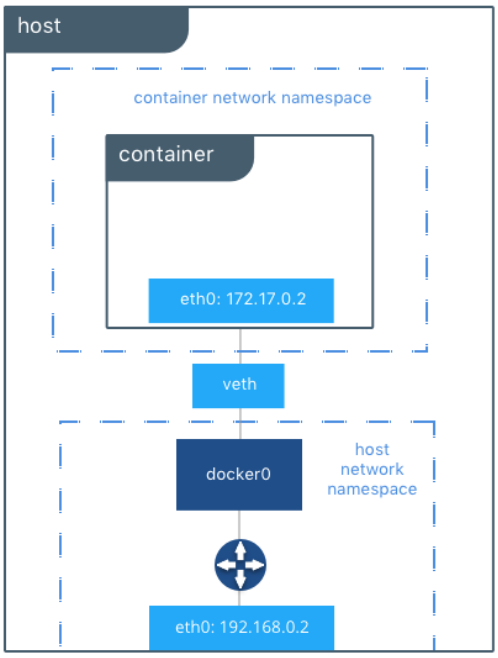
Można teraz przyjrzeć się konfiguracji tego trybu sieciowego.

```
student@student-PwCh0:~$ ip address | grep enp
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    inet 10.0.2.15/24 brd 10.0.2.255 scope global dynamic noprefixroute enp0s3
student@student-PwCh0:~$ docker exec c1 ip addr | grep enp
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP qlen 1000
    inet 10.0.2.15/24 brd 10.0.2.255 scope global dynamic enp0s3
student@student-PwCh0:~$ curl localhost
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
student@student-PwCh0:~$
```

Kontener „c1” oraz host macierzysty „widzą” te same interfejsy z tą samą konfiguracją. Poprzez interfejs enp0s3 można połączyć się z każdym kontenerem. Potwierdza to przedstawiony wynik działania polecenia „curl localhost”.

Default Docker Bridge Network

Ten tryb jest trybem domyślnym (default) w środowisku Docker. Podstawą działania tego trybu jest most (ang. bridge) tworzony na hoście macierzystym, który domyślnie nazywany jest docker0. Idea tego trybu jest przedstawiona na rysunku poniżej.



W tym trybie, kontener wykorzystuje dedykowaną mu przestrzeń nazw sieciowych a sposób (i możliwości) komunikacji z otoczeniem są zależne od tak ustawień sieciowych kontenera jak i parametrów mostu „docker0”.

Istnienie wymienionego mostu można sprawdzić np. w następujący sposób:

```
student@student-PwCh0:~$ 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: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 08:00:27:ba:7d:fc brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.15/24 brd 10.0.2.255 scope global dynamic noprefixroute enp0s3
       valid_lft 85379sec preferred_lft 85379sec
    inet6 fe80::69ea:78bb:304:a572/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default
    link/ether 02:42:14:85:38:5f 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
student@student-PwCh0:~$ brctl show
bridge name     bridge id               STP enabled     interfaces
docker0         8000.02421485385f       no
```

UWAGA: Aby skorzystać z drugiego z przedstawionych poleceń, w systemie macierzystym musi być zainstalowany pakiet „bridge-utils”. (można go doinstalować poleceniem $ sudo apt-get install bridge-utils).

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. W trakcie tworzenia kontenera w tym trybie można podać swój zakres adresów. Można również skonfigurować środowisko Docker by domyślnie wykorzystywało inny most (np. utworzony i skonfigurowany samodzielnie). Szczegóły tych i innych, bardziej zaawansowanych konfiguracji można znaleźć w dokumentacji Docker, np.

https://docs.docker.com/network/bridge/#disconnect-a-container-from-a-user-defined-bridge

Przykład wykorzystania trybu domyślnego mostu (Default Docker Bridge Network) jest przedstawiony poniżej.

```
student@student-PwCh0:~$ docker run -it --name default busybox:latest sh
/ # ip address
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue 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
4: eth0@if5: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue
    link/ether 02:42:ac:11:00:02 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.2/16 brd 172.17.255.255 scope global eth0
       valid_lft forever preferred_lft forever
/ # ip route
default via 172.17.0.1 dev eth0
172.17.0.0/16 dev eth0 scope link  src 172.17.0.2
/ # ping -c 2 ubuntu.com
PING ubuntu.com (91.189.94.40): 56 data bytes
64 bytes from 91.189.94.40: seq=0 ttl=61 time=47.157 ms
64 bytes from 91.189.94.40: seq=1 ttl=61 time=46.926 ms

--- ubuntu.com ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 46.926/47.041/47.157 ms
/ # exit
```

Jak widać na przytoczonym listingu, kontener „default” ma jeden interfejs sieciowy eth0, który stanowi wyjście trasy domyślnej. Dodatkowo, posiada on możliwość łączenia się z sieciami zewnętrznymi. Szczegółowe informacje na temat jego konfiguracji można uzyskać za pomocą poznanego już polecenia (oczywiście „default” to nazwa utworzonego wcześniej kontenera”:

$ docker inspect default

W sekcje konfiguracji ustawień sieciowych można potwierdzić informacje o konfiguracji trybu sieciowego. Przedstawia to zrzut ekranu poniżej (proszę pamiętać, że kontener musi być działający).


```
"NetworkSettings": {
    "Bridge": "",
    "SandboxID": "3ef1bb2be905596f704caed8d8abc5e61089eb9b9a8cbd5cfe207eea43fd85ab",
    "HairpinMode": false,
    "LinkLocalIPv6Address": "",
    "LinkLocalIPv6PrefixLen": 0,
    "Ports": {},
    "SandboxKey": "/var/run/docker/netns/3ef1bb2be905",
    "SecondaryIPAddresses": null,
    "SecondaryIPv6Addresses": null,
    "EndpointID": "bb73ebd2766524db3cad42415ea171668f71cb361cf8815b115c55894ed3dc52",
    "Gateway": "172.17.0.1",
    "GlobalIPv6Address": "",
    "GlobalIPv6PrefixLen": 0,
    "IPAddress": "172.17.0.2",
    "IPPrefixLen": 16,
    "IPv6Gateway": "",
    "MacAddress": "02:42:ac:11:00:02",
    "Networks": {
        "bridge": {
            "IPAMConfig": null,
            "Links": null,
            "Aliases": null,
            "NetworkID": "24751b39c0b329a7e65ea2e6d2176cdee56b0df8430d2e7df4e5a6cbfe71b27f",
            "EndpointID": "bb73ebd2766524db3cad42415ea171668f71cb361cf8815b115c55894ed3dc52",
            "Gateway": "172.17.0.1",
            "IPAddress": "172.17.0.2",
            "IPPrefixLen": 16,
            "IPv6Gateway": "",
            "GlobalIPv6Address": "",
            "GlobalIPv6PrefixLen": 0,
            "MacAddress": "02:42:ac:11:00:02",
            "DriverOpts": null
```


Skoro znana jest nazwa sieci (bridge) to można posłużyć się kolejnym poleceniem, aby poznać konfigurację tej sieci.

$ docker network inspect [options] <nazwa sieci>

W przypadku badanej, domyślnej sieci bridge, wynik polecenia jest następujący.

```
student@student-PwCh0:~$ docker network inspect bridge
[
    {
        "Name": "bridge",
        "Id": "24751b39c0b329a7e65ea2e6d2176cdee56b0df8430d2e7df4e5a6cbfe71b27f",
        "Created": "2018-10-25T17:59:19.021991777+02:00",
        "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": {
            "458397bb3794b5b4c567ceeb38b81c4908c3249d5001cc271e2587fd29dcf781": {
                "Name": "default",
                "EndpointID": "bb73ebd2766524db3cad42415ea171668f71cb361cf8815b115c55894ed3dc52",
                "MacAddress": "02:42:ac:11:00:02",
                "IPv4Address": "172.17.0.2/16",
                "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": "1500"
        },
        "Labels": {}
    }
]
```

Z poziomu hosta macierzystego, można sprawdzić, że do domyślnego mostu „docker0” dołączono wirtualny interfejs sieciowy utworzonego kontenera oraz, że pojawiła się trasa do puli adresowej, przypisanej do sieci „bridge”.

```
student@student-PwCh0:~$ brctl show
bridge name     bridge id               STP enabled     interfaces
docker0         8000.024253e416d3       no              vethcaf335b
student@student-PwCh0:~$ ip route
default via 10.0.2.2 dev enp0s3 proto dhcp metric 100
10.0.2.0/24 dev enp0s3 proto kernel scope link src 10.0.2.15 metric 100
169.254.0.0/16 dev enp0s3 scope link metric 1000
172.17.0.0/16 dev docker0 proto kernel scope link src 172.17.0.1
```

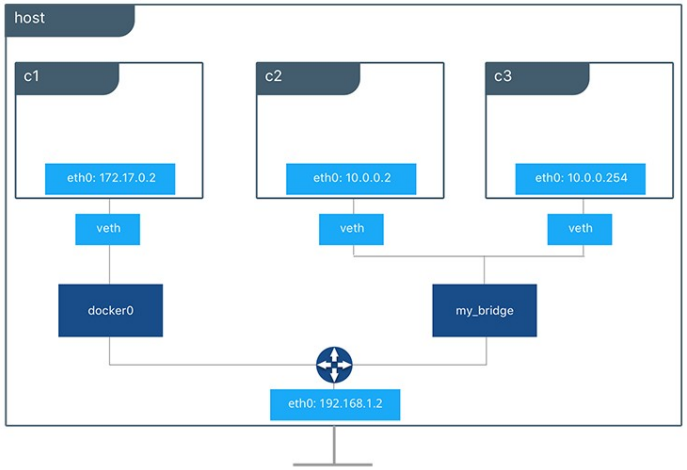
UWAGA: Należy pamiętać że domyślny tryb sieciowy jest jedynym, który w pełni wspiera legacy links. Dodatkowo, wykorzystując go nie ma możliwości korzystania z mechanizmów wykrywania usług na bazie nazw (komunikacja opiera się o numerację IP) oraz nie można nadawać punktom końcowym adresów spoza podanej wyżej puli.

User-Defined Bridge Networks

Obecnie najbardziej zalecanym trybem sieciowym jest tryb mostu definiowanego przez użytkownika. Tryb ten opiera się o możliwość definiowania w środowisku Docker własnych sieci dowolnego typu. Jednym z nich może być omawiany tryb mostu definiowanego przez użytkownika. Pośród jego wielu zalet, istotna jest możliwość ręcznego przypisywania adresów (pojedynczych adresów jak i puli adresów)

UWAGA: Jeśli w tym trybie nie podano zakresu adresów (adresu) punktów końcowych i sieci to wewnętrzny sterownik IPAM przypisuje je jako kolejną, nie wykorzystywaną podsieć (adres) z puli adresów prywatnych.

Idea konfiguracji połączeń sieciowych w tym trybie jest przedstawiona na rysunku poniżej.



Dla funkcjonowania tego trybu konieczne jest wykorzystanie dodatkowego mostu (innego niż domyślny „docker0”. Na rysunku jest on reprezentowany jako most o nazwie „my_bridge”.

Pierwszym krokiem jest utworzenie sieci. Służy do tego polecenie:

$ docker network create [options] <nazwa sieci>

```
student@student-PwCh0:~$ 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 copying 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
```

Dobrym zwyczajem jest deklarowanie przynajmniej adresacji podsieci wykorzystywanej przez ten tryb.

UWAGA: Zgodnie z poprzednią uwagą, w przypadku tego trybu, kwestia ewentualnego nakładania się pull adresowych należy do programisty. Wobec tego, jeśli nie korzysta się z mechanizmów IPAM to ewentualna pomyłka spowoduje błąd tworzenia sieci. Z drugiej strony, własne pule są niezwykle przydatne do panowania” nad większymi projektami.

Przykład utworzenia sieci „my_bridge” i jej parametry są przedstawione poniżej.

```
student@student-PwChO:~$ docker network create -d bridge --subnet 10.10.0.0/24 my_bridge
2f5a63ab46d316872e011be11d0d01b260bc1377b8e9a420488543f627019eb4
student@student-PwChO:~$ docker network inspect my_bridge
[
    {
        "Name": "my_bridge",
        "Id": "2f5a63ab46d316872e011be11d0d01b260bc1377b8e9a420488543f627019eb4",
        "Created": "2018-10-25T19:14:11.629232444+02:00",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "10.10.0.0/24"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {},
        "Options": {},
        "Labels": {}
    }
]
```

Kolejnym krokiem jest utworzenie kontenerów przyłączonych do utworzonej sieci. Niech będą to kontenery o nazwach „c2” oraz „c3” zgodnie z oznaczeniami na poprzednim rysunku.

```
student@student-PwChO:~$ docker run -itd --name c2 --net my_bridge busybox sh
d3cd5b0e726c306c592af13ae58be53f8f434364377fcd0555e9f17a90e2b95e
student@student-PwChO:~$ docker run -itd --name c3 --net my_bridge --ip 10.10.0.254 busybox sh
519addef4b45d68f0526660de3496f5227aaea4b04b48da121b257f3c670cc04
```

W przypadku drugiego kontenera (c3) adres został zdefiniowany w poleceniu uruchamiającym kontener. Dla kontenera c2 adres został przypisany automatycznie z puli adresów przynależnych do sieci „my_bridge”. Te stwierdzenia można sprawdzić jak poniżej, za pomocą analizy wyników polecenia „docker inspect” w odniesieniu do kontenerów c2 i c3.

Dla „c2”:

```
"Networks": {
    "my_bridge": {
        "IPAMConfig": null,
        "Links": null,
        "Aliases": [
            "a21072a5b7f1"
        ],
        "NetworkID": "2f5a63ab46d316872e011be11d0d01b260bc1377b8e9a420488543f627019eb4",
        "EndpointID": "9afa653109e53baf74772df24bcc12652f1a682e84ed69fdd907a3aab74bae62",
        "Gateway": "10.10.0.1",
        "IPAddress": "10.10.0.2",
        "IPPrefixLen": 24,
        "IPv6Gateway": "",
        "GlobalIPv6Address": "",
        "GlobalIPv6PrefixLen": 0,
        "MacAddress": "02:42:0a:0a:00:02",
        "DriverOpts": null
    }
```

Dla „c3”:

```
"Networks": {
    "my_bridge": {
        "IPAMConfig": {
            "IPv4Address": "10.10.0.254"
        },
        "Links": null,
        "Aliases": [
            "da6b43b5809b"
        ],
        "NetworkID": "2f5a63ab46d316872e011be11d0d01b260bc1377b8e9a420488543f627019eb4",
        "EndpointID": "ec7c3368d38a9e93fdf8e68c2cdd9f11497606ec9a951f84ac9bdabcc7857446",
        "Gateway": "10.10.0.1",
        "IPAddress": "10.10.0.254",
        "IPPrefixLen": 24,
        "IPv6Gateway": "",
        "GlobalIPv6Address": "",
        "GlobalIPv6PrefixLen": 0,
        "MacAddress": "02:42:0a:0a:00:fe",
        "DriverOpts": null
    }
```


Można też wykorzystać poznane wcześniej polecenia, informujące o sposobie konfiguracji tego trybu sieciowego.

```
student@student-PwChO:~$ brctl show
bridge name     bridge id               STP enabled     interfaces
br-2f5a63ab46d3         8000.0242b89bdbd7       no              veth328431c
                                                        veth9f51fcd
docker0         8000.024253e416d3       no              vethcaf335b
student@student-PwChO:~$ docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
24751b39c0b3        bridge              bridge              local
be91c7e8ade7        host                host                local
2f5a63ab46d3        my_bridge           bridge              local
3617fd744479        none                null                local
```

Dodatkowo, można dane poszczególnych mostów i interfejsów wirtualnych można sprwdzić standardowymi narzędziami systemowymi na hoście macierzystym.

```
student@student-PwChO:~$ ip address | grep br-2f5a63ab46d3
8: br-2f5a63ab46d3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    inet 10.10.0.1/24 brd 10.10.0.255 scope global br-2f5a63ab46d3
10: veth9f51fcd@if9: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master br-2f5a63ab46d3 state UP group default
12: veth328431c@if11: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master br-2f5a63ab46d3 state UP group default
```

UWAGA 1: 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.

UWAGA 2: Każdy kontener może być przyłączony do wielu sieci. Jednocześnie dany kontener może mieć tylko po jednym wirtualnym interfejsie sieciowym przyłączonym do danej sieci.

Do przyłączania i odłączania kontenera od sieci opartej o dany sterownik (pracującej w danym trybie sieciowym) służą dwa dedykowane polecenia:

$ docker network connect [options] <nazwa sieci> <nazwa/ID kontenera>

$ docker network disconnect [options] <nazwa sieci> <nazwa/ID kontenera>

Wskazane mechanizmy można zilustrować przez utworzenie kontenera o nazwie „c1” (nazwa zgodna z oznaczeniami z rysunku wyżej), który domyślnie (brak deklaracji trybu sieciowego) zostanie przyłączony do mostu „docker0” (tryb domyślnego mostu sieciowego). Następnie ten działający kontener przyłączymy do utworzonej wcześniej sieci „my_bridge”.

```
student@student-PwCh0:~$ docker run -itd --name c1 busybox sh
1902da847644d673d65256f1b912121e875749cf1a111bc0ce84638e35d866a7
student@student-PwCh0:~$ docker network connect my_bridge c1
student@student-PwCh0:~$ brctl show
bridge name     bridge id               STP enabled     interfaces
br-2f5a63ab46d3         8000.0242b89bdbd7       no              vetha5a8c8d
                                                        vethd3c552a
                                                        vethfdb7f3e
docker0         8000.024253e416d3       no              veth12c72f8
```

Sprawdzając konfigurację mostów na hoście macierzystym widać, że do mostu „docker0” jest podłączony jeden wirtualny interfejs kontenera (pochodzi z kontenera „c1”) a do mostu „br-2f5a63ab46d3” są podłączone trzy wirtualne interfejsy (należące po jednym do kontenerów „c1”, „c2” oraz „c3”). Dodatkowo, można wyświetlić konfigurację interfejsów w kontenerze „c1” i tym samym potwierdzić, że należy od do dwóch sieci, odpowiednio „bridge” oraz „my_bridge”.

```
student@student-PwCh0:~$ docker exec c1 ip address
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue 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
17: eth0@if18: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue
    link/ether 02:42:ac:11:00:02 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.2/16 brd 172.17.255.255 scope global eth0
       valid_lft forever preferred_lft forever
19: eth1@if20: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue
    link/ether 02:42:0a:0a:00:03 brd ff:ff:ff:ff:ff:ff
    inet 10.10.0.3/24 brd 10.10.0.255 scope global eth1
       valid_lft forever preferred_lft forever
```

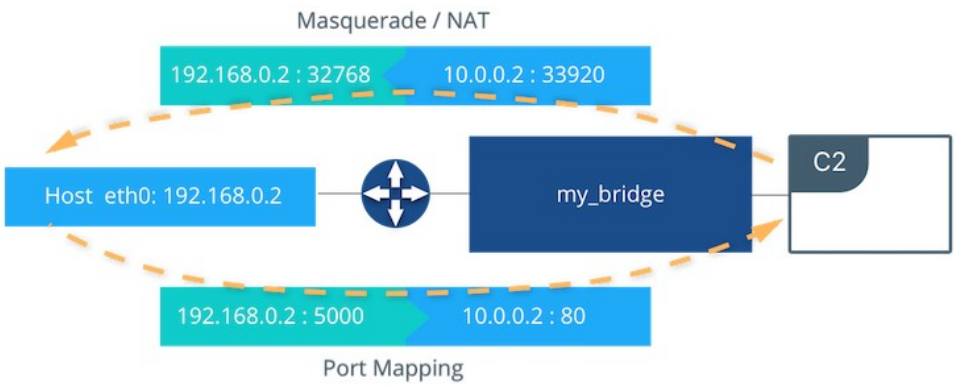
Analogiczne potwierdzenia można uzyskać analizując wyniki działania poleceń „docker inspect” oraz „docker network inspect” w odniesieniu do poszczególnych kontenerów i sieci.

Dostęp do kontenerów z sieci zewnętrznej

W domyślnym ustawieniu, wszystkie kontenery podłączone do danej sieci mogą komunikować się za pośrednictwem wszystkich portów sieciowych. Z kolei komunikacja pomiędzy sieciami oraz z sieci zewnętrznych (ang. ingress traffic) podlega blokowaniu na firewall-u. Z tego powodu należy ręcznie określić czy i jaki zakres dostępu z zewnątrz do aplikacji pracujących w kontenerach, ma być możliwy. Realizowane to jest poprzez znany mechanizm mapowania portów (internal port mapping). Docker musi opublikować porty wystawione (ang. exposed) na interfejsach hosta macierzystego i mapować je na porty wirtualnych interfejsów kontenera.

UWAGA: Z przyczyn oczywistych, ten mechanizm nie funkcjonuje dla sieci wykorzystującej Host Network Driver.

Ideę tego mechanizmu ilustruje rysunek poniżej.



Górna strzałka symbolizuje ruch ruch egress z kontenera „c2”. Jest on udostępniony domyślnie w oparciu o mechanizm NAT (ruch jest obsługiwany przez masquerade/SNAT na wolny port na interfejsie hosta – typowo zakres wykorzystywanych portów to 32768 – 60999).

Dolna strzałka symbolizuje komunikację ingress, która musi być skonfigurowana zgodnie z potrzebami aplikacji/usługi pracującej w kontenerze. Polega to ma zmapowaniu portu na interfejsie hosta na wymagany port na wirtualnym interfejsie kontenera. Służy temu opcja -p (--publish) lub –P podawana w poleceniu „docker run”.

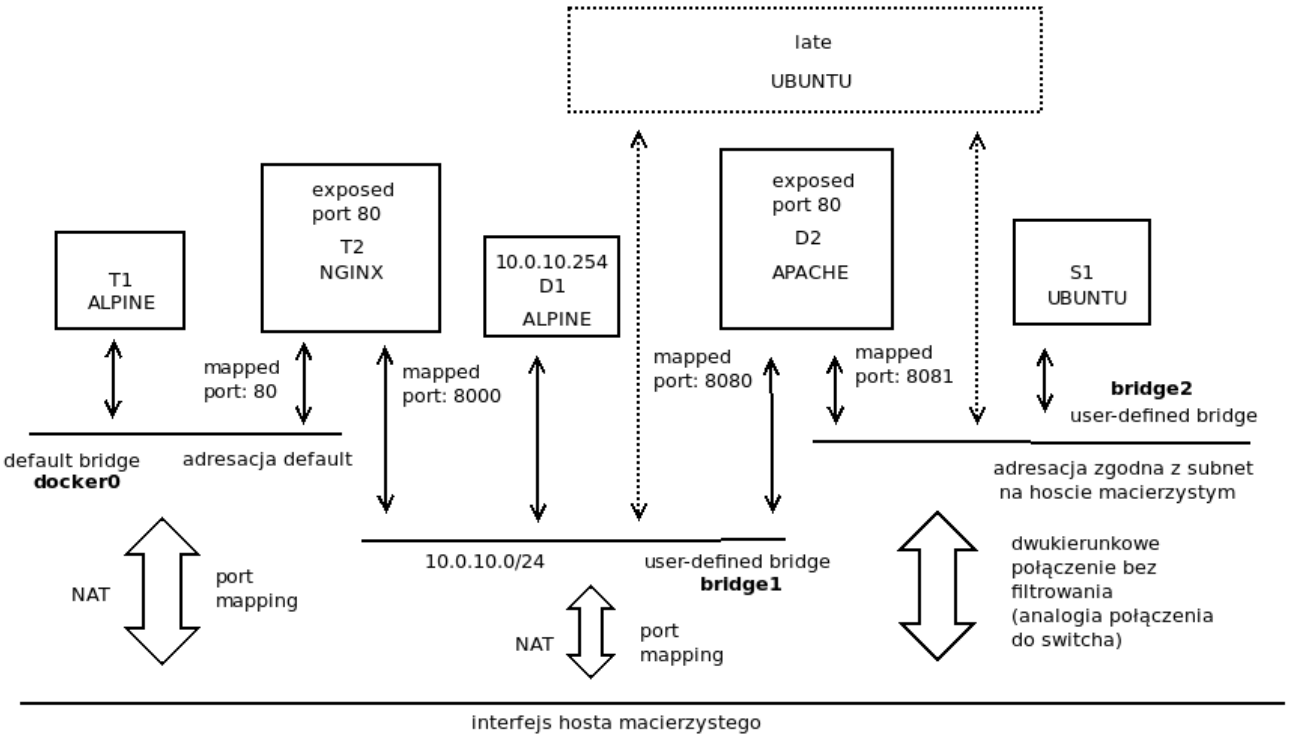
UWAGA: Opcja -P mapuje wszystkie „exposed” porty danego kontenera na losowe porty na interfejsie hosta macierzystego dlatego większą kontrolę daje opcja -p, która pozwala na jawne tworzenie zmapowanych par portów.

```
$ docker run -d --name C2 --net my_bridge -p 5000:80 nginx
```

Informacje na temat w jaki sposób i które porty są mapowane można uzyskać za pomocą polecenia.

```
student@student-PwCh0:~$ docker port --help

Usage:  docker port CONTAINER [PRIVATE_PORT[/PROTO]]

List port mappings or a specific mapping for the container
```

3Z1. Należy zbudować następującą strukturę połączeń pomiędzy kontenerami i siecią zewnętrzną.



Konfiguracja powinna zostać opisana jako skrypt bash z komentarzami. Dodatkowe uwagi do skryptu:

1. Kontenery „T2” oraz „D2” nasłuchują na wewnętrznych portach 80 (portach swoich wirtualnych interfejsów sieciowych). Porty mają być mapowane na podane porty na interfejsie hosta macierzystego. Proszę zwracać uwagę, że mapowanie jest realizowane indywidualnie dla każdego endpointa (interfejsu wirtualnego kontenera) przyłączonego do danej sieci.

2. Kontener na bazie obrazu Ubuntu (np. latest) „late” ma zostać przyłączony do dwóch sieci pracujących pod nadzorem User-defined Network Driver. Powinno być to wykonane jako ostatnie działanie. Aby uniknąć domyślnego przyłączenia do mostu Docker0, można utworzyć ten kontener (docker create) a następnie przyłączyć do wymaganych sieci (docker network connect). Na końcu można go uruchomić.

2. Kontenery „D1” oraz „S1” mają być przyłączone do sieci wykorzystującej User-defined Network Driver. Należy uzyskać możliwość niefiltrowanej (bez ograniczeń) i dwukierunkowej komunikacji z interfejsem hosta macierzystego. Odpowiada to sytuacji, jakby wymienione kontenery i host macierzysty przyłączone były do przełącznika sieciowego (można też takie połączenie porównać do trybu sieciowego bridge w Virtualbox). Należy przyjrzeć się ustawieniom iptables dlo tego trybu i w opracowywanym skrypcie umieścić odpowiednie reguły (ewentualnie usunąć wybrane reguły) tak by możliwa była taka komunikacja.

Pomocny link: https://docs.docker.com/v17.09/ - sekcja „Configure networking”

W sprawozdaniu, oprócz wymienionego wyżej skryptu, należy umieścić końcową konfigurację wszystkich sieci, tablicę routingu na hoście macierzystym oraz kontenerach „T2” oraz „D2”, jak również istotne fragmenty konfiguracji iptables. W ocenie końcowej brane będzie pod uwagę wskazanie miejsc w w/w listingach, które potwierdzają poprawność wykonania zadania.

Pytania:
1. Kontenery „D2” oraz „S1” mają mieć bezpośredni dostęp (via most) do interfejsu hosta macierzystego.

a) W jaki sposób ustalić i skonfigurować pulę adresową dla podsieci na moście „bridge2” ?

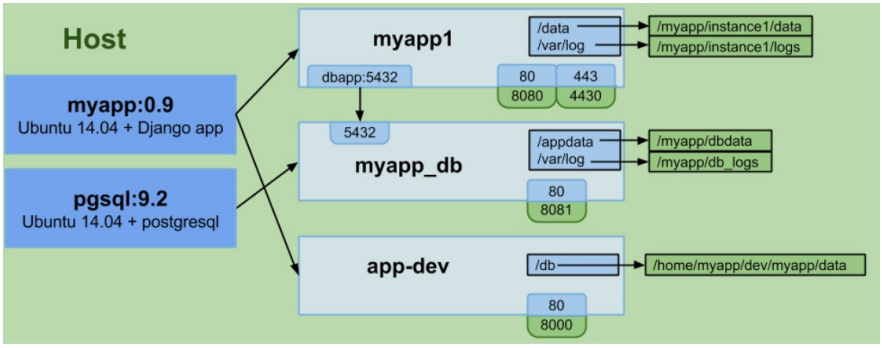
b) Czy kontenery „D2” i „S1” mogą wykorzystać mechanizm dynamicznego przypisywania adresów (DHCP) jeśli w tym segmencie sieci dostępny byłby serwer DHCP. Czy wymaga to dodatkowej konfiguracje któryś z elementów zbudowanej infrastruktury ? Jeśli tak, to proszę podać co i jak należałoby skonfigurować.

B. Łącza w środowisku Docker

Zgodnie z informacjami na wstępie, linki jako forma komunikacji pomiędzy kontenerami Docker jest traktowana jako rozwiązanie stare (ang. legacy). Tym niemniej jest ono czasem nadal stosowane szczególnie w trybie domyślnego mostu, gdzie nie można korzystać z mechanizmów wykrywania usług opartych o nazwy. Ich drugą, główną zaletą jest możliwość prostego przenoszenia zmiennych środowiskowych.

UWAGA: Obecnie rozszerzana jest możliwość wymiany i zdalnej konfiguracji zmiennych środowiskowych w oparciu o User-defined Network Driver. Ten tryb jest zatem zalecany zawsze gdy w nowych projektach potrzebne jest skorzystanie z własności oferowanych przez tradycyjne linki.

W celu realizacji komunikacji na bazie linków pomiędzy dwoma kontenerami Docker na jednym hoście stworzono możliwość odwoływania się do istniejących kontenerów w momencie uruchamiania nowego kontenera. Taki nowo-utworzony kontener Docker ma dodany do swojej konfiguracji alias (o nazwie określonej podczas jego tworzenia). Nowy kontener oraz kontener, na który wskazuje link mogą się komunikować (są zlinkowane). Ideę wykorzystania linków w środowisku Docker ilustruje rysunek poniżej:



Załóżmy, że kontener *myapp_db* jest już uruchomiony. Można teraz utworzyć kontener zawierający serwer HTTP (na rysunku kontener *myapp1*) i zdeklarować dla niego link do kontenera z bazą danych (*myapp_db*). Link ten należy nazwać, w przykładzie otrzymał on nazwę dbapp. Po uruchomieniu kontenera *myapp1*, aby skomunikować się z bazą danych wystarczy użyć nazwy hostname dbapp. Środowisko Docker wymaga określenia, który port (numer portu) będzie oferowany na danym kontenerze dla innych kontenerów w celu utworzenia linku. Gdy nowy kontener utworzy łącze do kontenera oferującego dany port to środowisko Docker automatycznie wygeneruje zestaw zmiennych wewnątrz tego nowego kontenera wraz z informacją o porcie, wybranym dla potrzeb funkcjonowania linku (w przykładzie z rysunku powyżej jest to port 5432).

Załóżmy, że dwa kontenery Docker na tym samym hoście, przyłączone do mostu „docker0”, muszą wymieniać się danymi w celu realizacji określonej usługi. Niestety, kontenery Docker nie rozgłaszają swoich adresów IP innym kontenerom (stosowana jest dynamiczna konfiguracja interfejsów). W celu zbudowania połączenia należy przykładowo wydać następujące polecenia.

```
student@student-PwCh0:~$ docker run -itd --name myapp_db -p 8000:8000 busybox sh
bffd751975ca56475530eefe3f8c7a45aadf5862753d0b3177b1f51c2636bcd2
student@student-PwCh0:~$ docker run -itd --name myapp2 --link myapp_db:mylink busybox sh
dcfab35cf1ef9f3005437bf68d280577a606965373d07fd64bc7cc6bdf91c950
```

```
student@student-PwCh0:~$ docker attach myapp2
/ # ping -c 2 mylink
PING mylink (172.17.0.2): 56 data bytes
64 bytes from 172.17.0.2: seq=0 ttl=64 time=0.109 ms
64 bytes from 172.17.0.2: seq=1 ttl=64 time=0.178 ms

--- mylink ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 0.109/0.143/0.178 ms
/ # cat /etc/hosts
127.0.0.1       localhost
::1     localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
172.17.0.2      mylink bffd751975ca myapp_db
172.17.0.3      dcfab35cf1ef
/ # printenv
MYLINK_PORT_8000_TCP_PORT=8000
MYLINK_PORT_8000_TCP_PROTO=tcp
HOSTNAME=dcfab35cf1ef
SHLVL=1
HOME=/root
MYLINK_PORT_8000_TCP=tcp://172.17.0.2:8000
MYLINK_PORT=tcp://172.17.0.2:8000
MYLINK_NAME=/myapp2/mylink
TERM=xterm
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
PWD=/
MYLINK_PORT_8000_TCP_ADDR=172.17.0.2
/ #
```

Pierwszy kontener „myapp_db” wystawił port 8000, który jest mapowany na port zewnętrzny również 8000. Uruchamiając kolejny kontener „myapp2” zdeklarowany został link do kontenera „myapp_db” o nazwie „mylink”. Kontener „myapp2” może teraz łączyć się po nazwie tego linku z „myapp_db” (bez znajomości jego adresu IP). Utworzenie linku spowodowało odpowiednią

modyfikację pliku *hosts* w katalogu etc kontenera „myapp2”. Kontener ten otrzymał również nowe zmienne środowiskowe. Ilustruje to zrzut ekranowy powyżej.

3Z2. W poprzednim zadaniu utworzone były dwa kontenery „T1” oraz „T2” przyłączone do mostu doker (domyślny tryb sieciowy). Proszę ręcznie utworzyć kontener „T2” a potem uruchomić „T1” zlinkowany do „T1”. W sprawozdaniu proszę podać użyte polecenia oraz zawartość pliku hosts i zmienne systemowe na kontenerze „T1”.

Pytania:

1. Czy link jest dwukierunkowy, tj czy na bazie utworzonego linku można komunikować się (np. pingować) z kontenera „T2” na „T1”. Odpowiedź proszę uzasadnić.

2. Czy w pozostałych trybach tj. w sieciach wykorzystujących User-defined Network Driver oraz Host Network Driver jest możliwe tworzenie linków. Jeśli tak, zaprezentuj przykładową budowę linku, jeśli nie to uzasadnij odpowiedź.

3. Czy możliwe jest tworzenie linków pomiędzy sieciami skonfigurowanymi w różnych trybach sieciowych. Odpowiedź uzasadnij.

C. Wykorzystując domyślny tryb sieciowy mostu (docker0), kontenery mogą wzajemnie się komunikować po adresach IP. Stosowanie nazw jako celu (np. w poleceniu ping) nie jest możliwe poza wykorzystaniem omówionego wyżej mechanizmu linków. Natomiast w sieci wykorzystującej tryb mostu definiowanego przez użytkownika możliwe jest połączenie wykorzystujące nazwy poszczególnych kontenerów. Sieć ta pozwala na wykorzystanie jeszcze jednej metody nazywanej aliasami o zasięgu sieci (ang. network-scoped aliases). Dzięki nim, można odwoływać się do poszczególnych kontenerów przyłączonych do danej sieci poprzez zdefiniowane aliasy (inne niż nazwa kontenera). Aliasy można deklarować podczas uruchamiania kontenera w danej sieci w trybie mostu definiowanego przez użytkownika lub przyłączając działający kontener do tej sieci. Przykładowe polecenia:

$ docker run -itd - - name example - - net my_bridge - -network-alias first busybox sh

Tym poleceniem uruchomiony został kontener o nazwie „example”, przyłączony jest do sieci „my_bridge” i posiada alias o zasięgu całej sieci o nazwie „first”

$ docker network connect - - alias first my_bridge example

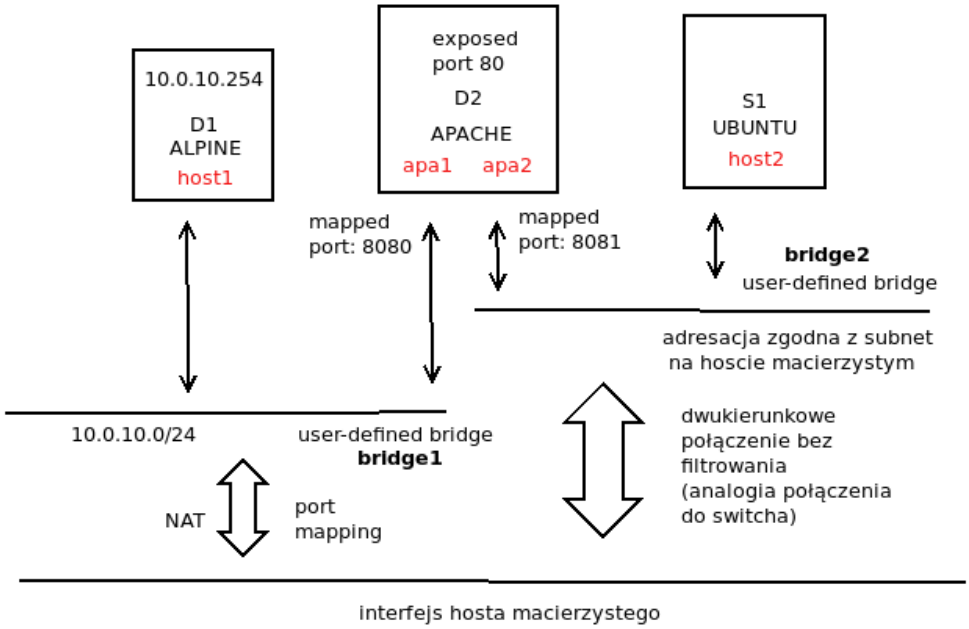
Z kolei to polecenie daje ten sam efekt co poprzednie, pod warunkiem, że kontener „example” działa lub jest stworzony.

3Z3. Zmodyfikuj skrypt z zadania 3Z1 tak by uwzględniał on użycie aliasów (aliasy są podane na rysunku za pomocą nazw w kolorze czerwonym). Skrypt w nowej postaci powinien zawierać niezbędne komentarze. Jego treść należy umieścić w sprawozdaniu.

Pytania:

1. Czy można używać aliasów do komunikacji pomiędzy kontenerami przyłączonymi do dwóch różnych sieci ale pracujących w trybie mostu definiowanego przez użytkownika (np. pomiędzy host1 a host2) ? Odpowiedź uzasadnij powołując się na sposób implementacji aliasów o zasięgu sieci.

2. W przypadku pomyślnej konfiguracji w zadaniu 3Z1 sieci wykorzystującej most „bridge2” sprawdź czy możliwe jest zatem korzystanie ze zdefiniowanych aliasów na hoście macierzystym przy połączeniach do kontenera „S1” lub „D2” ? Uzasadnij otrzymany rezultat.



Zestawienie poleceń wykorzystywanych przy podstawowej pracy z trybami sieciowymi oraz przykłady ich użycia można znaleźć pod adresem: https://docs.docker.com/engine/reference/commandline/network/ oraz kolejnych podrozdziałach poświęconych poszczególnym poleceniom.