LABORATORIUM PROGRAMOWANIA W CHMURACH OBLICZENIOWYCH

LABORATORIUM NR 4.

Wykorzystanie magazynów przechowywania danych w środowisku Docker.

Niniejsze laboratorium głównie poświęcone jest zasadom korzystania z zewnętrznych wolumenów danych w celu przechowywania danych wykorzystywanych przez kontenery Docker. Dodatkowo, poruszone są też kwestie, których zrozumienie jest niezbędne do tworzenia poprawnych konfiguracji plików Dockerfile, co będzie tematem kolejnych zajęć.

A. Rola zmiennych środowiskowych. Na poprzednim laboratorium podkreślane było, że wykorzystanie mechanizmu linków do realizacji połączeń jednokierunkowych pomiędzy kontenerami, pozwala również na przekazywanie parametrów do środowiska kontenera inicjalizującego link. W przypadku korzystania z trybu sieci mostkowych definiowanych przez użytkownika (bez linków) przekazywanie parametrów musi być realizowany w odmienny sposób. Jest to istotne zagadnienie, ponieważ zmienne środowiskowe nie dotyczą tylko kwestii konfiguracji komunikacji pomiędzy kontenerami.

Zazwyczaj odbywa się poprzez właściwą konfigurację aplikacji (najrzadsza metoda) przeznaczonej do pracy w kontenerze, w trakcie definiowania polecenia uruchamiającego kontener (często stosowana ale głównie do modyfikacji/przykrywania" zmiennych zdefiniowanych w Dockerfile) lub poprzez odpowiednie wpisy w wspomnianym pliku Dockerfile.

Jednocześnie, właściwa realizacji przekazywania parametrów ułatwia (bądź umożliwia) tworzenie bardziej złożonych struktur. Poniżej zilustrowane to jest przykładem (rozwiązanie oparte o przekazywanie parametrów za pośrednictwem konfiguracji aplikacji i w oparciu o poznane własności trybów sieciowych), w którym w systemie potrzebujemy umieścić za dwie aplikacji korzystające z dwóch instancji tego samego serwera (wykorzystywany jest serwer redis). Dodatkowo aplikacje te mają być „widoczne" z sieci zewnętrznej.

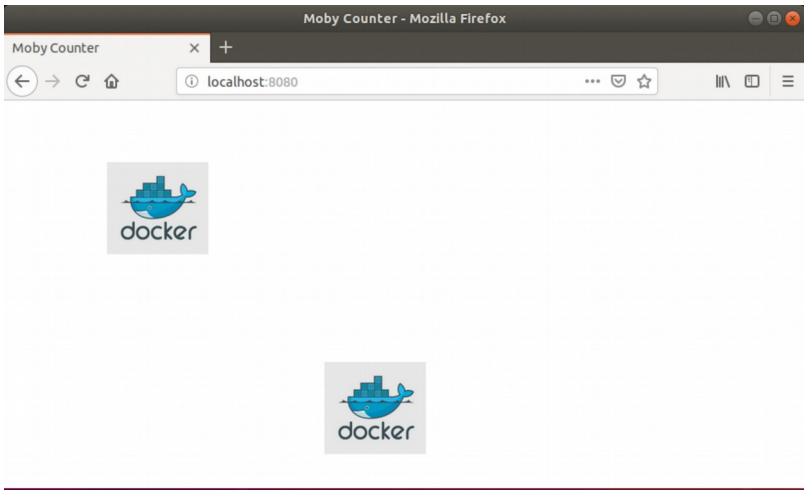
Oczywistym podejściem jest umieszczenie ich w dwóch sieciach mostkowych definiowanych przez użytkownika. Utworzmy pierwszą sieć o nazwie „moby0"

```
student@student-PwCh0:~$ docker network create moby0
21a5d2ac72ae41cc5c635608a4452d7650bc6a9d79b01e4a3bacc2dcd21cb5d7
student@student-PwCh0:~$ docker network inspect moby0
[
    {
        "Name": "moby0",
        "Id": "21a5d2ac72ae41cc5c635608a4452d7650bc6a9d79b01e4a3bacc2dcd21cb5d7",
        "Created": "2018-11-07T23:36:35.963739079+01:00",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "172.19.0.0/16",
                    "Gateway": "172.19.0.1"
                }
            ]
        },
```

W tej sieci umieśćmy dwa kontenery, odpowiednio: serwer redis oraz aplikacje webową (dostępna w DockerHub pod nazwą: russmckendrick/moby-counter), która korzysta z serwera redis (komunikuje się z nim po nazwie) i która ma „wystawiony” port zewnętrzny 8080.

```
student@student-PwChO:~$ docker run -d --name redis --network moby0 redis:alpine
0f361dd21fe3bb9852c9a2e2f21355e906464bfaef5c862bb3baa4a485246987
student@student-PwChO:~$ docker run -d --name couner --network moby0 -p 8080:80 russmcke
ndrick/moby-counter
Unable to find image 'russmckendrick/moby-counter:latest' locally
latest: Pulling from russmckendrick/moby-counter
ff3a5c916c92: Pull complete
0384617ecf25: Pull complete
3e2743173da8: Pull complete
40c2a5cd7772: Pull complete
e00657f4abd2: Pull complete
32312bfbca18: Pull complete
Digest: sha256:d0f51203130cb934a2910c2e0d577e68b7ab17962ce01918d37d7de9686553cc
Status: Downloaded newer image for russmckendrick/moby-counter:latest
df66c5cf1f0e667b15838ae7273a0b11750b456de2bd50d60581ce2aa8a47a43
student@student-PwChO:~$
```

Aplikację moby-counter (program reaguje na każde kliknięcie wyświetleniem logo dockera) można uruchomić w przeglądarce, na systemie macierzystym.



Aplikacja moby-counter posiada deklarację zmiennych, które są przekazywane do środowiska Docker i które umożliwiają komunikację pomiędzy tą aplikacją a serwerem redis po jego nazwie i na wskazany port. Jest to konfiguracja zmiennych przekazywana z poziomu konfiguracji aplikacji, wygodniej jest umieszczać ją w samym pliku Dockerfile.

```
var port = opts.redis_port || process.env.USE_REDIS_PORT || 6379
var host = opts.redis_host || process.env.USE_REDIS_HOST || 'redis'
```

Poprawność takiej konfiguracji można potwierdzić jak poniżej:

```
student@student-PwChO:~$ docker container exec couner ping -c 2 redis
PING redis (172.19.0.2): 56 data bytes
64 bytes from 172.19.0.2: seq=0 ttl=64 time=0.073 ms
64 bytes from 172.19.0.2: seq=1 ttl=64 time=0.097 ms

--- redis ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 0.073/0.085/0.097 ms
student@student-PwChO:~$
```

Zanim można przejść do implementacji drugiej, bliźniaczej aplikacji w systemie , trzeba sprawdzić na jakiej zasadzie odbywa się komunikacji pomiędzy mobu-counter a serwerem redis. W tym celu przyjrzyjmy się plikom w kontenerze „couner” co ilustruje zrzut ekranu poniżej. Widać, ze komunikacja nie jest zrealizowana w oparciu o rozwiązywanie nazw oparte o plik */etc/hosts* (tak było w przypadku linków), natomiast w systemie kontenera skonfigurowany został wewnętrzny (lokalny) serwer DNS (zdeklarowany w pliku */etc/resolv.conf*)/

```
student@student-PwChO:~$ docker container exec couner 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
172.19.0.3      df66c5cf1f0e
student@student-PwChO:~$ docker container exec couner cat /etc/resolv.conf
nameserver 127.0.0.11
options ndots:0
```

Proste zapytanie o serwer redis z poziomu kontenera „couner” powoduje zwrócenie adresu IP i nazwy kontenera z serwerem redis w sieci moby0.

```
student@student-PwChO:~$ docker container exec couner nslookup redis 127.0.0.11
Server:     127.0.0.11
Address 1: 127.0.0.11

Name:       redis
Address 1: 172.19.0.2 redis.moby0
```

Teraz tworzymy drugą sieć, np. o nazwie moby 1 i podłączamy drugą instancję aplikacji moby-counter.

```
student@student-PwChO:~$ docker network create moby1
d277d5a2852457f175ec005bc98f1c173c3b63d00f5b37fba690b67005cd6c8c
student@student-PwChO:~$ docker run -itd --name couner2 --network moby1 -p 9090:80 russmckendrick/moby-counter
e97a6c593bedd621c8cd3af252a96010b4206c7258b3eb3f3f87f88f4c9a0917
```

Nie mozemy podłączyć tej drugiej aplikacji do poprzedniej sieci tj. moby0, ponieważ posiada ana ten sam zestaw parametrów środowiskowych co jej pierwsza (już działająca instancja). Oczywiste jest (na podstawie poprzedniego laboratorium), że kontener „couner2” nie będzie „widział” serwera redis pomimo że korzysta z tego samego, lokalnego DNS-a.

```
student@student-PwChO:~$ docker container exec couner2 ping -c 2 redis
ping: bad address 'redis'
student@student-PwChO:~$ docker container exec couner2 cat /etc/resolv.conf
nameserver 127.0.0.11
options ndots:0
```

Z powyższego zrzutu ekranowego wynika, że nie możemy zainstalować w sieci moby1 kolejnego serwera/kontenera o nazwie redis bo taki już istnieje w sieci moby0 a system DNS nie może mieś dwóch kontenerów o tej samej nazwie. Z drugiej strony, konfiguracja zmiennych srodowiskowych, zawarta w konfiguracji aplikacji moby-counter skłoniła nas do korzystania z dwóch sieci i wymusza komunikację po nazwie, właśnie redis. Rozwiazaniem tego problemu jest wykorzystanie poznanych na poprzednim laboratorium, mechanizmu aliasów o zasięgu sieci.

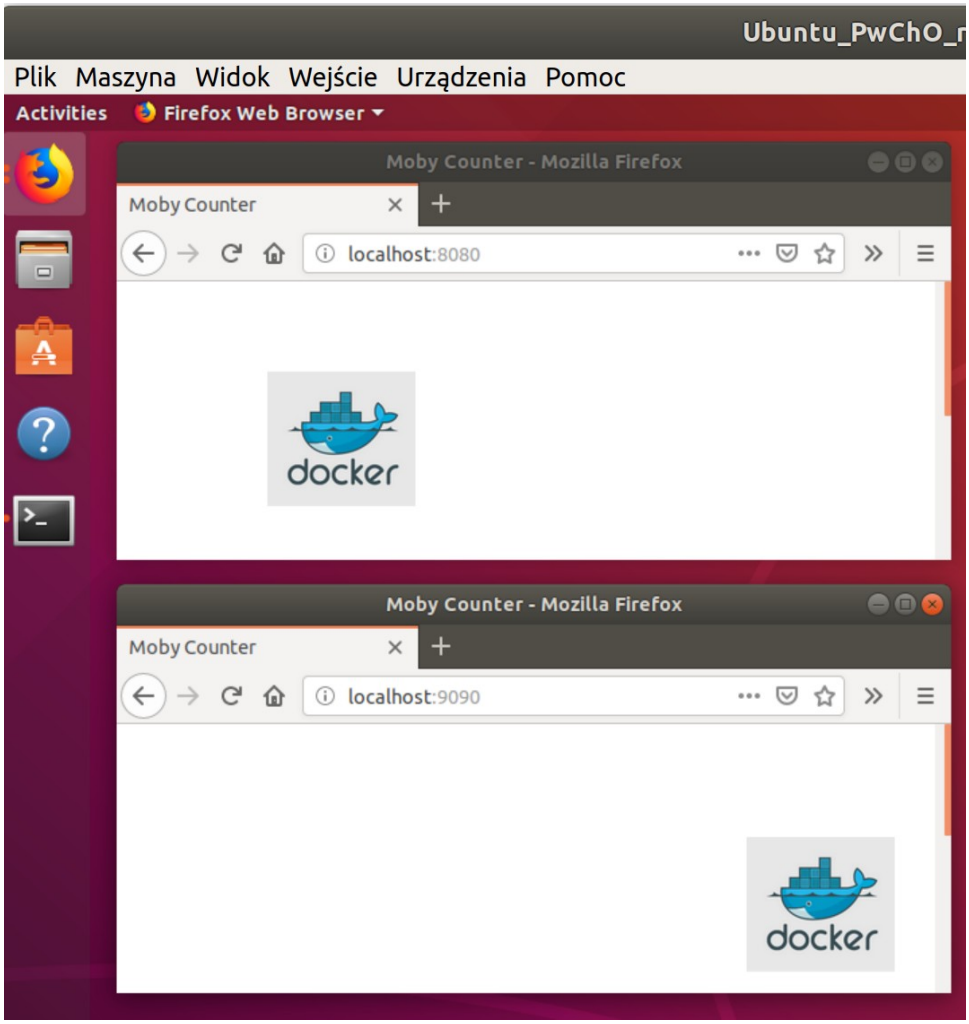
```
student@student-PwChO:~$ docker container run -d --name redis1 --network moby1 --network-alias redis redis:alpine
```

Dzięki temu, z poziomu kontenera „couner2” w sieci moby2 „widoczny” jest serwer redis a DNS działa poprawnie.

```
student@student-PwChO:~$ docker exec couner2 nslookup redis 127.0.0.11
Server:    127.0.0.11
Address 1: 127.0.0.11

Name:      redis
Address 1: 172.20.0.3 redis1.moby1
```

W efekcie, w systemie mamy działające dwie kopie tej samej aplikacji (moby-counter) komunikujące się zgodnie z konfiguracją zmiennych srodowiskowych w tej aplikacji z dwoma serwerami redis w oparciu o jeden, lokalny DNS implementowany w przypadku korzystania z wielu sieci mostkowych, definiowanych przez użytkownika). Można sprawdzić, ze obie aplikacje są dostępne „z zewnątzr” na portach, odpowiednio 8080 oraz 9090.

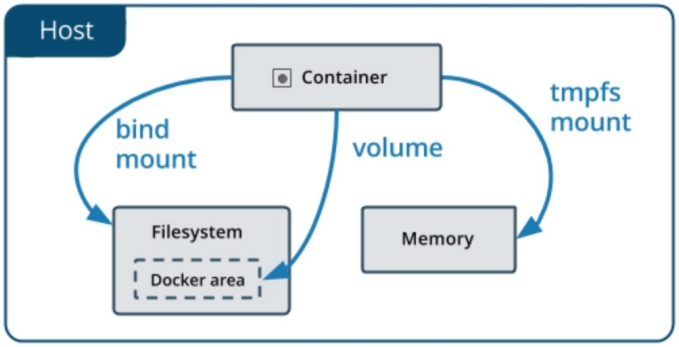


Jak napisano na wstępie, ten prosty przykład ilustruje związek pomiędzy deklaracjami zmiennych a organizacją i architekturą komunikacji w środowisku Docker. Pozostałe metody przekazywania zmiennych środowiskowych zawarte są w zadaniu dodatkowym (na końce tej instrukcji) oraz będą omawiane na kolejnych ćwiczeniach.

B. Przechowywanie danych w kontenerach, zgodnie z informacjami ze wcześniejszego laboratorium, jest możliwe za pomocą najwyzszej warstwy stosu systemu plików kontenera (warstwa RW). Należy jednak pamiętać, że w domyślnej konfiguracji, kontenery Docker są ulotne co oznacza, że, działają tak długo jak długo wykonywane jest zadanie (działa aplikacja) zlecona do wykonania w kontenerze. Po jej zakończeniu, dane zapisane w tej warstwie systemu plików kontenera są bezpowrotnie tracone.

UWAGA: Po zatrzymaniu kontenera, również dostęp do danych zapisanych w wewnętrznym systemie plików jest niemożliwy.

W wielu przypadkach konieczne jest jednak, by aplikacja współdzieliła dane (oraz dostęp do danych) lub by dane nie były usuwane po zakończeniu działania kontenera. Przykładem mogą być bazy danych, dane dla potrzeb serwisów webowych czy też logi aplikacji. Dodatkowo, w większości przypadków nie jest wskazane (czy nawet możliwe) by te zasoby były umieszczane wewnątrz obrazu Dockera np. ze względu na znaczne zwiększenie wymaganych zasobów wykorzystywanych przez kontener. Z tego względu, w środowisku Docker dostępny jest tzw. mechanizm wolumenów. Istnieją trzy, podstawowe metody realizacji tej funkcji, co ilustruje rysunek poniżej.



Volumes są przechowywane w systemie plików maszyny macierzystej (hosta, na którym zainstalowano i skonfigurowano środowisko Docker). Zarządzaniem predefiniowaną pulą na wolumeny zajmuje się środowisko Docker.

UWAGA: Domyślnym miejscem przechowywania wolumenów są podkatalogi w katalogu */var/lib/docker/volumes/* (wymagane prawa root-a)

Wolumeny są podstawową i zalecaną metodą przechowywania danych permanentnych.

Bind mounts są wolumenami, których położenie w systemie plików systemu macierzystego określa sam użytkownik.

UWAGA 1. Funkcjonowanie tych wolumenów podlega wszelkim prawom, jakie obowiązują w przypadku korzystania z unix-owego narzędzia mount (np. przykrywanie zawartości katalogu/pliku zasobami mountowanymi itd.)

UWAGA 2. Inne procesy, działające w systemie macierzystym, ktore mają prawa dostępu do wolumenów typu “bind mounts” mogą wpływać na ich zawartość I konfigurację. Należy to brać pod uwagę, przy podejmowaniu decyzji o wykorzystywaniu tych wolumenów.

tmpfs mounts są metodą przechowywania danych wyłącznie w pamięci systemu macierzystego (nie są nigdy zapisywane w systemie plików). W związku z tym ich zastosowanie ogranicza się do ewentualnego przechowywania danych o charakterze tymczasowym oraz do sytuacji gdy czas dostępy (zapis/odczyt) do danych jest czynnikiem krytycznym dla działania danej aplikacji (oferują najnizzsze opóźnienia w dostępie do danych).

Ostatnia z implementacji wolumenów nie będzie przedmiotem dalszych ćwiczeń. Dokumentację korzystania z tmpfs mounts można znaleźć pod adresem: https://docs.docker.com/storage/tmpfs/

W przypadku wolumenów, istnieje kilka metod ich deklarowania i wykorzystania. Poniżej przedstawione są przyj]kłady najczęściej wykorzystywanych schematów postępowania.

1. **Tworzenie wolumenów permanentnych**. Środowisko Docker CE dostarcza polecenia do tworzenia volumenów, które nie muszą być wstępnie skojarzone z żadnym kontenerem. Przykładowo, można utworzyć wolumen *TestVol1* za pomocą polecenia jak poniżej:

```
student@student-PwChO:~$ docker volume create --name TestVol1
TestVol1
student@student-PwChO:~$ docker volume ls
DRIVER              VOLUME NAME
local               0db35580223299b5aee608518a37d15383ebf58e285abfc42e02bf74a8cbf605
local               1aa86a187d30078adc282bb15390149410b8c93bbf8d3a82771593d0fbd93d50
local               TestVol1
local               e0ff37726ae7efb12d60889433e22bdfc3242c5f97ad5ccb48bce31a1505edc5
```

Z kolei listę istniejących wulumenów uzyskuje się poprzez polecenie

$ docker volume ls

UWAGA. Jeżeli nie poda się nazwy wolumenu, to środowisko docker samodzielnie utworzy dla niego nazwę. Liczba i nazwy wolumenów, przedstawione na powyzszym zrzucie ekranu mogą się różnić w zależności od wcześniej uruchamianych kontenerów.

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

UWAGA. Proszę KONIECZNIE zapoznać się z fragmentem dokumentacji. Dostępnym pod adresem: https://docs.docker.com/storage/volumes/#choose-the--v-or---mount-flag . 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).

Podobnie jak w przypadku sieci czy kontenerów, dla wolumenów istnieje możliwość skorzystania z polecenia inspect i wyświetlenia ustawień konkretnego wolumenu. Dla uprzednio utworzonego wolumeny „TestVol1”

```
student@student-PwCh0:~$ docker volume inspect TestVol1
[
    {
        "CreatedAt": "2018-11-08T22:15:08+01:00",
        "Driver": "local",
        "Labels": {},
        "Mountpoint": "/var/lib/docker/volumes/TestVol1/_data",
        "Name": "TestVol1",
        "Options": {},
        "Scope": "local"
    }
]
```

Teraz ten wolumen można przyłączyć do danego kontenera podczas jego uruchamiania. Składnia opcji –mount wymaga podania pary kluczy, source oraz terget:

source=<nazwa wolumen>, target=<:miejsce mountowania w systemie plików kontenera>

UWAGA: Ścieżka w kluczu „target” powinna być ścieżką bezwzględną (zaczynać się od /). Jeżeli podana ścieżka nie istnieje to zostanie ona dodana do obrazu w wyniku działania polecenia *docker run*

Informacje o podłączonym wolumenie można odnaleźć poleceniem inspect, wyświetlającym konfigurację uruchomionego kontenera (w tym wypadku był to kontener „voltest”). Informacje są zawarte w sekcji „mounts”.

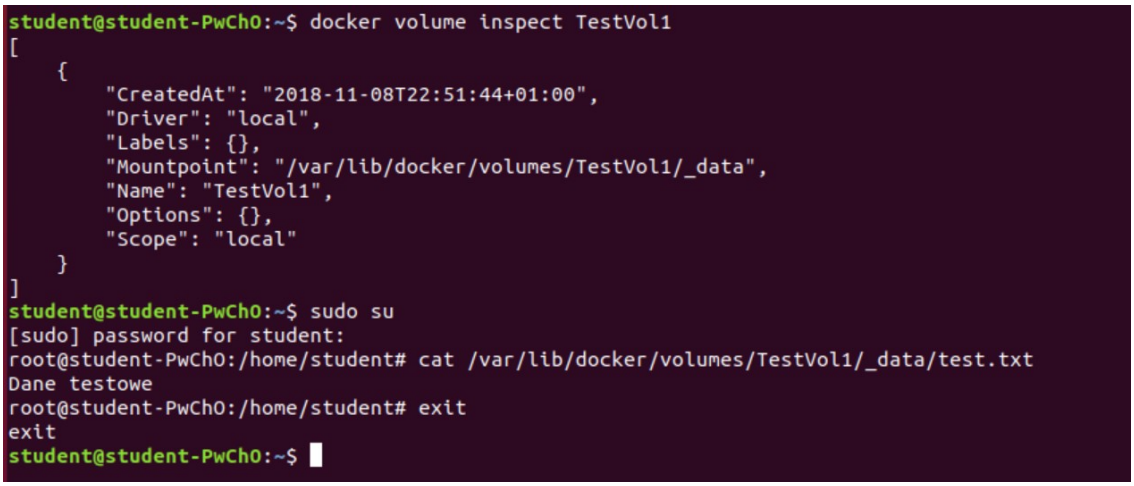
```
student@student-PwCh0:~$ docker run -ti --rm --name voltest --mount source=TestVol1,target=/magazyn ubuntu:latest
Unable to find image 'ubuntu:latest' locally
latest: Pulling from library/ubuntu
473ede7ed136: Pull complete
c46b5fa4d940: Pull complete
93ae3df89c92: Pull complete
6b1eed27cade: Pull complete
Digest: sha256:29934af957c53004d7fb6340139880d23fb1952505a15d69a03af0d1418878cb
Status: Downloaded newer image for ubuntu:latest
root@224c6e75bfc0:/# ls -l
total 68
drwxr-xr-x   2 root root 4096 Oct 18 21:03 bin
drwxr-xr-x   2 root root 4096 Apr 24  2018 boot
drwxr-xr-x   5 root root  360 Nov  8 21:36 dev
drwxr-xr-x   1 root root 4096 Nov  8 21:36 etc
drwxr-xr-x   2 root root 4096 Apr 24  2018 home
drwxr-xr-x   8 root root 4096 Oct 18 21:02 lib
drwxr-xr-x   2 root root 4096 Oct 18 21:02 lib64
drwxr-xr-x   2 root root 4096 Nov  8 21:15 magazyn
drwxr-xr-x   2 root root 4096 Oct 18 21:02 media
drwxr-xr-x   2 root root 4096 Oct 18 21:02 mnt
drwxr-xr-x   2 root root 4096 Oct 18 21:02 opt
dr-xr-xr-x 214 root root    0 Nov  8 21:36 proc
drwx------   2 root root 4096 Oct 18 21:03 root
drwxr-xr-x   1 root root 4096 Oct 19 00:47 run
drwxr-xr-x   1 root root 4096 Oct 19 00:47 sbin
drwxr-xr-x   2 root root 4096 Oct 18 21:02 srv
dr-xr-xr-x  13 root root    0 Nov  8 21:36 sys
drwxrwxrwt   2 root root 4096 Oct 18 21:03 tmp
drwxr-xr-x   1 root root 4096 Oct 18 21:02 usr
drwxr-xr-x   1 root root 4096 Oct 18 21:03 var
root@224c6e75bfc0:/#
```

Zgodnie ze zrzutem ekranowym powyzej, po wykonaniu polecenia jesteśmy “wewnątrz” kontenera a kataloh magazym został utworzony Skoro tak, to można zapisać jakieś przykładowe dane do kontenera:

```
echo „Dane testowe” > /magazyn/test.txt
```

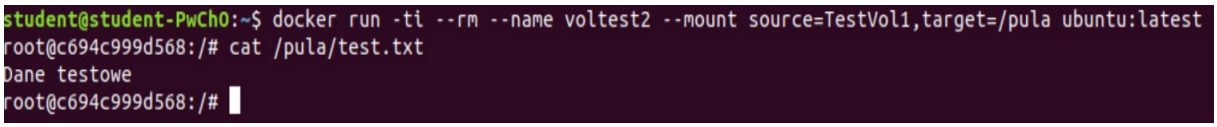
Ponieważ przy uruchomieniu kontenera użyta została opcja *–rm* to po jego wyłączeniu dane powinny zostać utracone. Po wydaniu polecenia *exit* można

sprawdzić czy wolumen *TestVol1* jest nadal obecny. Służy do tego ponownie polecenie docker volume inspect



Jak widać na powyzszym zrzucie ekranowym, wolumen istnieje po zakończeniu pracy kontenera a zawarte w nim dane są dostępne z poziomu systemu macierzystego

Teraz można utworzyć kolejny kontener i podmontować do niego wolumen *TestVol1*.



Jak widać na powyzszym rysunku, wulemen został podmontowany a dane są dostępne z poziomu kontenera.

2. Permanentne wolumeny tworzone wraz z kontenerem. Zgodnie z tym co napisano na wstępie, 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.

3. Wykorzystywanie deklaracji wolumenów w plikach Dockerfile. W wielu plikach Dockerfile zawarte są wpisy definiujące wolumeny. Jako przykład użyty zostanie obraz serwera redis. Zanim jednak przejdziemy do dalszych przykładów, dobrze będzie usunąć wszystkie kontenery i wolumeny.

UWAGA: Nie można usunąć wolumenu, który jest powiązany z działającym (lub zatrzymanym) kontenerem. Dlatego najpierw należy zatrzymać i usunąć kontener a dopiero potem wolumen.

Usunięcie volumenów odbywa się za pomocą polecenia:

$ docker volume rm <nazwa wolumenu>

Można też użyć polecenia:

$ docker volume prune

które usuwa wszystkie wolumeny (z uwzględnieniem uwagi powyżej).

Na DockerHub dostępny jest obraz oraz dokumentacja serwera redis https://hub.docker.com/_/redis/

Po wejściu na ten adres, w oknie „Full Description” jest podana lista dostępnych obrazów. Dla wersja 5.0.0 (latest) należy „kliknąć na link do Dockerfil-a. Na końcu zawartości tego Dockerfile można odnaleźć wpisy definiujące „wolume” oraz „workdir” tak jak ilustruje to rysunek poniżej.

```
68          \
69          make -C /usr/src/redis -j "$(nproc)"; \
70          make -C /usr/src/redis install; \
71          \
72          rm -r /usr/src/redis; \
73          \
74          apt-get purge -y --auto-remove $buildDeps
75
76  RUN mkdir /data && chown redis:redis /data
77  VOLUME /data
78  WORKDIR /data
79
80  COPY docker-entrypoint.sh /usr/local/bin/
81  ENTRYPOINT ["docker-entrypoint.sh"]
82
83  EXPOSE 6379
84  CMD ["redis-server"]
```

Powyższe zapisy oznaczają, że uruchamiając kontener z serwerem redis, tworzony jest dedykowany, anonimowy (nazwa ustalona automatycznie przez system) volumen, który podmontowywany jest w systemie plików kontenera w ścieżce „/data”

4Z1. Proszę zweryfikować wykorzystanie wolumenów, według dwóch poniższych zadań:

A.
- Proszę utworzyć kontener na bazie obrazu ubuntu:latest wraz z wolumenem *FirstVol*,. Kontener ma za zadanie zapisać do pliku *lista.txt* nazwy wszystkich plików zawierających literę ***a*** z katalogu /etc kontenera. Plik *lista.txt* powinien być umieszczony w katalogu, w którym podmontowano wolumin *FirstVol* a kontener po wykonaniu tej pracy powinien zakończyć działanie.
- Proszę uruchomić kolejny kontener na bazie ubuntu:latest z podłączonym wolumenem *FirstVol*. Kontener powinien wykonać zadanie polegające na zliczeniu ilości słów w pliku *lista.txt* (plik powinien być dostępny na katalogu, w którym zamontowano ten wolumen) a następnie wyświetleniu tej liczby.

Skrypt wraz z niezbędnymi komentarzami i ilustrację jego działania należy umieścić w sprawozdaniu.

B. Proszę utworzyć własny, nowy kontener o nazwie RedisRob. Następnie uruchomić serwer redis w wersji 5.0.0 (latest) z wykorzystaniem tego wolumeny zamiast anonimowego kontenera tak by on właśnie był podnontowany w katalogu /data systemu plików serwera. Poprawność konfiguracji proszę potwierdzić wynikiem działania odpowiedniego polecenia. Użyte polecenia i wynik działania proszę umieścić w sprawozdaniu.

C. Współdzielenie wolumenów. Poprzednie przykłady zakładały podłączanie wolumenu do pojedynczego kontenera. W praktyce często zdarza się, że wiele Dockerów powinno korzystać z tego samego wolumenu.

UWAGA. Przedstawione poniżej polecenia wykorzystują alternatywną (w stosunku do opcji - - mount) składnię opartą o opcję -v

Krok 1: Utwórzmy kontener o nazwie Docker1 z przyłączonym wolumenem DVol1:

$ docker run -ti --name=D1 -v DVol1:/data1 ubuntu:latest

Następnie, stwórzmy na zamontowanym wolumenie plik *test1.txt* z zawartością w postaci tekstu „Plik wspołdzielony przez wiele dokerow" i wyłączmy kontener (polecenie *exit*).

Krok 2: Utwórzmy kolejny kontener (wykorzystując kolejne okno terminala), tym razem o nazwie D2 i połączmy do niego wolumen utworzony w poprzednim kroku. Należy w tym celu wydać polecenie:

$ docker run -ti --name=D2 -v Dvol1:/data2 ubuntu:latest

Ponieważ jesteśmy w terminalu dockera wobec tego można sprawdzić czy jest dostępny plik o nazwie *test.txt* i czy zawiera zapisane wcześniej dane (zdanie).

cat /data2/test.txt

Dopiszmy do tego pliku kolejne zdanie (kolejny wiersz) o treści: „Tu byłem – Docker2". Na koniec należy wyłączyć kontener.

Krok 3: Dość prostą metodą ominięcia problemów z ewentualnym, równoczesnym zapisem danych jest montowanie woluminów współdzielonych w trybie tylko do odczytu (read only). Utwórzmy kolejny doker o nazwie D3 i przetestujmy tą metodę. Należy użyć następującego polecenia:

$ docker run -tid –name=D3 --volumes-from D1:ro ubuntu:latest

Ponownie, należy sprawdzić czy plik test.txt jest dostępny i czy dostęp jest tylko w trybie odczytu (brak możliwości zmiany zawartości czy też usunięcia tego pliku).

UWAGA 1: W tym kroku wykorzystano opcje „- - volumes-from”. Pozwala ona na automatyczne podmontowanie wolumenu wykorzystywanego przez inny kontener o podanej nazwie oraz zachowanie nazwy katalogu w systemie plików kontenera, gdzie pojawi się ten wolumen.

UWAGA 2: Opcja „- - volumes-from” jest w większości przypadków wykorzystywana do tworzenia backup-ów wolumenów. Proszę KONIECZNIE zapoznać się z dokumentacją pod adresem:
https://docs.docker.com/storage/volumes/#backup-restore-or-migrate-data-volumes

Krok 4: Na koniec proszę usunąć wszystkie kontenery oraz wolumen *DVol1*

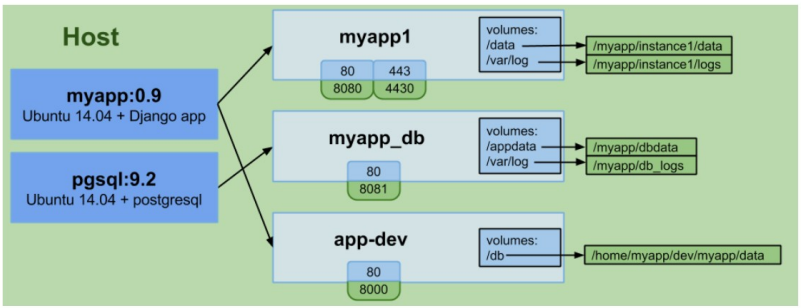
$ docker rm D1 D2 D3

$ docker volume rm DVol1

4Z2. Wykorzystując umiejętności z jednego z poprzednich ćwiczeń, proszę opracować możliwie najprostsze pliki Dockerfile dla 3 serwerów WWW na bazie serwera Apache.
- Należy też przygotować własną, prostą, statyczną stronę html. Pierwszy Dockerfile powinien pozwolić na stworzenie kontenera Docker z serwerem WWW, który wyświetla zawartość pliku html umieszczonego na utworzonym, zewnętrznym wolumenie.
- Kolejne dwa serwery WWW powinny korzystać z tego (utworzonego dla I serwera) wolumenu z tym, że pierwszy z nich z prawami „read-write” a drugi z prawami „read-only). Oba mają wyświetlać tą samą stronę.

Wszystkie pliki Dockerfile, polecenia i wynik ich działania wraz z niezbędnymi komentarzami, należy umieścić w sprawozdaniu. UWAGA: PROSZĘ UŻYWAĆ WYŁĄCZNIE OPCJI „--mount” a nie „-v”.

D. Wolumeny typu bind mounts. Istnieje również możliwość współdzielenia wolumenów w systemem plików systemu macierzystego.



By to zilustrować, wykorzystany będzie obraz serwera nginx. Poniższe polecenie tworzy katalog *nginxlogs* w katalogu domowym danego użytkownika (w naszym przypadku student) systemu macierzystego oraz skojarzony z nim volumen zamontowany (bind mount) w katalogu */var/log/nginx* kontenera.

UWAGA: 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.

```
student@student-PwChO:~$ mkdir ~/ngxlog
student@student-PwChO:~$ docker run -d --name ngx --mount type=bind,source=/home/student/ngxlog,target=/var/log/nginx -p5000:80 nginx
4e9ba7a76e98b7f5224e647bdcd9e4d1fa98f5351b1f754066d1879d2e52595a
student@student-PwChO:~$
```

Z serwerem nginx działającym w uruchomionym kontenerze można połączyć się za pomocą przeglądarki w systemie macierzystym. Kontener „nasłuchuje” na porcie 5000 i mapuje go na port 80, na którym skonfigurowany jest serwer nginx.

Teraz można sprawdzić, że w obu wymienionych wyżej katalogach, tak na systemie macierzystym jak i w kontenerze dostępne są logi serwera nginx.

```
student@student-PwChO:~$ ls ~/ngxlog/
access.log  error.log
student@student-PwChO:~$ docker exec ngx ls /var/log/nginx
access.log
error.log
student@student-PwChO:~$
```

4Z3. Uruchom trzy serwery nginx w taki sposób by:
- wszystkie serwery były podłączone do jednej sieci mostkowej definiowanej przez użytkownika,
- wszystkie serwery były dostępne z sieci zewnętrznej,
- poszczególne serwery zapisywały logi w trzech dedykowanych podkatalogach katalogu domowego na systemie macierzystym.

W sprawozdaniu proszę umieścić wszystkie użyte polecenia wraz z wynikiem ich działania oraz dowód, że logi serwerów zostały poprawnie zapisane i są dostępne w systemie macierzystym.

4Z4. Na jednym z poprzednich ćwiczeń instalowany był prywatny rejestr (registry) na podstawie obrazu https://hub.docker.com/_/registry/ . Proszę zapoznać się z zawarto-

ścią Dockerfile dla najnowszej wersji tego repozytorium. Na bazie tego tego obrazu proszę uruchomić rejestr w taki sposób by:
- był on dostępny pod adresem 10.0.100.2 oraz portem 6677,
- obrazy były przechowywane w katalogu /obrazy_priv w systemie macierzystym

W sprawozdaniu proszę umieścić dowód na poprawne wykonanie tego zadania.

D. Deklaracje korzystania z zasobów sprzętowych przez kontenery

Domyślnie, środowisko Docker nie zawiera ograniczeń na zasoby wykorzystywane przez kontenery. Innymi słowami, kontenery mogą współdzielić całość zasobów udostępnianych przez jądro systemu macierzystego. Wprowadzenie zmian i/lub ograniczeń możliwe jest podczas tworzenia lub uruchamiania kontenerów (polecenia: *create* oraz *run*). Deklaracje umieszczone w wymienionych poleceniach odnoszą się głównie do sposobu wykorzystania pamięci i procesora. Wszystkie opcje powiązane z tymi deklaracjami wraz ze szczegółowym opisem i przykładami ich użycia można odnaleźć pod adresem: https://docs.docker.com/config/containers/resource_constraints/

Uwaga: W systemach Ubuntu czy też Debian konieczne jest włączenie obsługi limitu dla pamięci swap. Włączenie tej opcji powoduje spodek wydajności środowiska Docker wobec czego zalecane jest by zmieniać te ustawienia wyłącznie w przypadku uzasadnionej potrzeby. Szczegóły można odnaleźć pod adresem: https://docs.docker.com/install/linux/linux-postinstall/#your-kernel-does-not-support-cgroup-swap-limit-capabilities

Zmianę ustawienia parametrów można sprawdzić w oparciu o poznane polecenie *inspect.* Istnieje również kolejne, przydatne polecenie

$ docker stats [opcje] [nazwa/ID kontenera]

Wyświetla ono podstawowe informacje o wykorzystywanych zasobach i ustawionych limitach w trybie „na żywo”.

Istnieje też możliwość graficznej reprezentacji ustawień środowiska i wykorzystywanych zasobów dzięki aplikacji cAdvisor (Container Advisor) opracowanej przez Google. Wszystkie szczegóły związane z jego uruchomieniem I wykorzystaniem można znaleźć ma GitHub pod odresem: https://github.com/google/cadvisor

E. Interfejs graficzny do zarządzania i monitorowania środowiska Docker CE

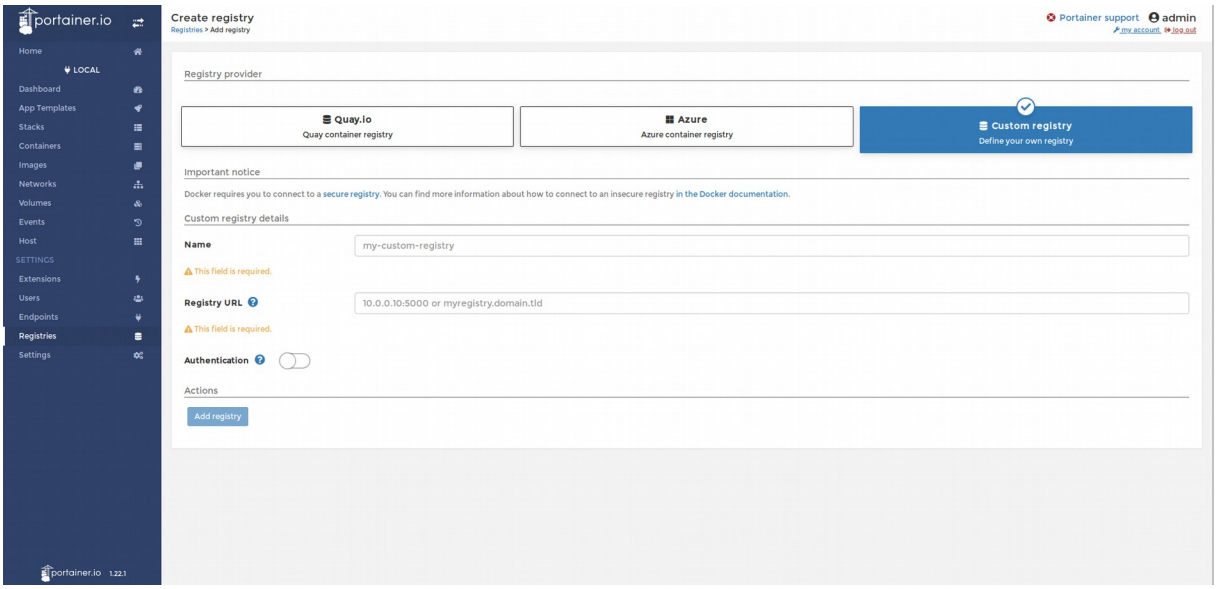
Poznane mechanizmy i narzędzia w środowisku Docker CE (jak i te, które przedstawione zostaną na kolejnych laboratoriach) dają możliwość ich wykorzystania z poziomu CLI. Istnieje również możliwość wykorzystania interfejsu GUI do zarządzania i monitorowania środowiska Docker. Co prawda, sam Docker CE nie dostarcza takiego rozwiązania ale istnieje wiele aplikacji wypełniających tą lukę. Obecnie jednym z najpopularniejszych rozwiązań jest Portainer. Dostępny jest on w postaci kontenera Docker na DockerHub. Z kolei dokumentacja Portainera jest dostępna pod adresem: https://portainer.readthedocs.io/en/latest/index.html

4Z5. Wykorzystując informację z podpunktu D proszę wprowadzić modyfikację w wykorzystaniu zasobów przez kontener o nazwie *limiter* utworzonego na podstawie najnowszej wersji obrazu alpine. Modyfikacje te mają polegać na:
1. ograniczeniu wykorzystywanej pamięci RAM do 512 MB,
2. ograniczenie działania kontenera do tylko tylko jednego rdzenia.
W sprawozdaniu, wykorzystując poznane narzędzia do monitorowania środowiska Docker (tekstowe jak i graficzne), należy umieścić zrzuty ekranów, które potwierdzają wprowadzenie w życie wymienionych wyżej ograniczeń.

4D. Dostęp z zewnątrz do prywatnego rejestry wymaga konfiguracji metody weryfikacji użytkownika. Aby to zrealizować, należy podczas uruchamiania kontenera, zawrzeć opcje odpowiedzialne za przekazywanie właściwych wartości zmiennych środowiskowych. Jest to druga z metod przekazywania tych zmiennych, z trzech wymienionych na wstępie instrukcji. Trzecia (oparta o Dockerfile będzie omówiona na kolejnych laboratoriach).

1. Proszę zapoznać się z dokumentację wdrażania prywatnego rejestru, która znajduje się pod adresem: https://docs.docker.com/registry/deploying/ a szczególnie sekcją https://docs.docker.com/registry/deploying/#native-basic-auth . Proszę uzupełnić rozwiązanie zadania 4Z4 o możliwość kontroli dostępu poprzez standardowy mechanizm htpasswd. W sprawozdaniu proszę umieścić opis kolejno wykonanych konfiguracji.

2. W aplikacji Portainer istnieje możliwość podłączenia własnego Registry. Odpowiednią stronę konfiguracji tego skladnika środowiska Docker w Portainer przedstawia rysunek poniżej.



Jakie działania konfiguracyjne utworzonego w punkcie 1 Registry są konieczne by móc z niego skorzystać w aplikacji Portainer. W sprawozdaniu proszę w punktach przedstawić wykonane działania.