


Sprawozdanie z laboratorium 3

Osoba wykonująca	Grupa	Data
Konrad Mazur	7.2/4	17.11.2020r.
Uczelnia	Wydział	Kierunek
Politechnika Lubelska 	Elektrotechniki i Informatyki 	Informatyka I. stopnia, stacjonarne
Temat laboratorium		
Wykorzystanie magazynów przechowywania danych w środowisku Docker.		

4Z1. Proszę utworzyć własny, nowy wolumen o nazwie RedisRob. Następnie uruchomić serwer redis w wersji latest z wykorzystaniem tego wolumenu zamiast anonimowego wolumenu tj. aby był on podmontowany w katalogu /data systemu plików kontenera (serwera). Poprawność konfiguracji proszę potwierdzić wynikiem działania odpowiedniego polecenia. Użyte polecenia i wynik działania proszę umieścić w sprawozdaniu.

Zadanie komendami:

Wyciągnij redis

docker pull redis – wyciąga wersję latest serwera redis

Stwórz volumen

```
docker volume create RedisRob
```

Odpal nowy kontener z podłączonym volumenem

docker run -it --rm --name myredis --mount source=RedisRob,target=/data redis – po stworzeniu kontenera i przełączeniu się do niego mamy:



```
student@PwChO-VB: ~
1:C 18 Nov 2020 18:10:31.673 # 000000000000 Redis is starting 000000000000
1:C 18 Nov 2020 18:10:31.673 # Redis version=6.0.9, bits=64, commit=00000000, modified=0, pid=1, just started
1:C 18 Nov 2020 18:10:31.673 # Warning: no config file specified, using the default config. In order to specify a config file use redis-server /path/to/redis.conf

Redis 6.0.9 (00000000/0) 64 bit
Running in standalone mode
Port: 6379
PID: 1

http://redis.io

1:M 18 Nov 2020 18:10:31.677 # Server initialized
1:M 18 Nov 2020 18:10:31.677 # WARNING overcommit_memory is set to 0! Background save may fall under low memory condition. To fix this issue add 'vm.overcommit_memory = 1' to /etc/sysctl.conf and then reboot or run the command 'sysctl vm.overcommit_memory=1' for this to take effect.
```

Wychodzimy i sprawdzamy zawartość wolumenu

```
root@PwChO-VB:/var/lib/docker/volumes# cd RedisRob/_data
root@PwChO-VB:/var/lib/docker/volumes/RedisRob/_data# ls
dump.rdb
```

Obecność pliku dump.rdb oznacza poprawne przypięcie wolumenu.

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 1 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”.

Stworzyłem wpierw volumen zad2

```
docker volume create zad2
```

Zrobiłem obrazy na podstawie dockerfile z poprzednich zajęć:

Strona to też kalka strony z poprzednich laboratoriów, czyli czas na świecie...

Dockerfile dla pierwszego kontenera

```
FROM scratch
ADD alpine-minirootfs-3.12.1-x86_64.tar.gz /
COPY strona.php /app/strona.php
LABEL maintainer="PwChO <konrad.mazur1@pollub.edu.pl>"
LABEL description="Zadanie2."
ENV PHPVERSION=7
RUN apk add --update apache2
RUN rm -rf /var/cache/apk/
RUN rm -rf /var/www/localhost/htdocs/index.html
RUN cat /app/strona.php > /var/www/localhost/htdocs/index.php
VOLUME /var/www/localhost/htdocs/
WORKDIR /var/www/localhost/htdocs/
EXPOSE 80/tcp
CMD ["/bin/sh"]
ENTRYPOINT ["httpd"]
CMD ["-D", "FOREGROUND"]
```

Dodane rozkazy na zielono.

Stworzyłem obraz.

```
docker build -t stapache .
```

I na podstawie obrazu kontener z zamontowanym volumenem

```
docker run -t -d -P --rm --name first --mount
source=zad2,target=/var/www/localhost/htdocs/ stapache
```

Dockerfile dla dwóch kolejnych

```
FROM scratch
ADD alpine-minirootfs-3.12.1-x86_64.tar.gz /
LABEL maintainer="PwChO <konrad.mazur1@pollub.edu.pl>"
LABEL description="Zadanie2."
ENV PHPVERSION=7
```

```
RUN apk add --update apache2
```

```
RUN rm -rf /var/cache/apk/
```

```
RUN rm -rf /var/www/localhost/htdocs/index.html
```

```
VOLUME /var/www/localhost/htdocs/
```

```
WORKDIR /var/www/localhost/htdocs/
```

```
EXPOSE 80/tcp
```

```
CMD ["/bin/sh"]
```

```
ENTRYPOINT ["httpd"]
```

```
CMD ["-D", "FOREGROUND"]
```

Powyższy dockerfile nie kopiuje ani nie zapisuje strony, lecz jak stworzymy kontener na pdst. Nowego obrazu

```
docker build -t ndapache .
```

```
docker run -t -d -P --rm --name second --volumes-from first:rw ndapache
```

To mamy kontener, który po odpaleniu strony o podanym porcie wyświetla to samo co pierwszy kontener

trzeci tworzymy za pomocą

```
docker run -t -d -P --rm --name third --volumes-from first:ro ndapache
```

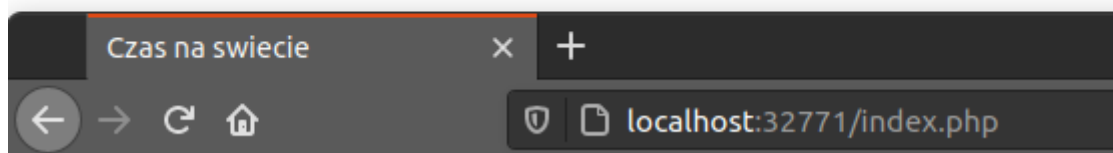
Sprawdzenie:

```
student@PwCh0-VB:~/lab4/zad2$ docker ps
```

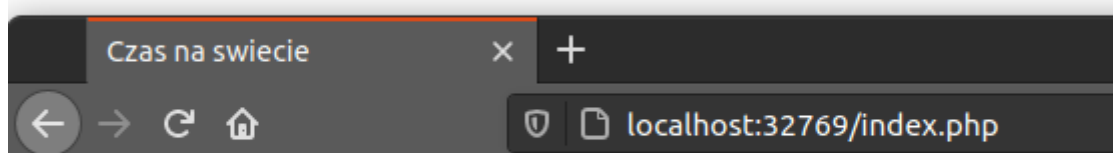
CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
a00d9f8745bf	ndapache	"httpd -D FOREGROUND"	38 seconds ago	Up 37 seconds	0.0.0.0:32771->80/tcp	third
967e08df8118	ndapache	"httpd -D FOREGROUND"	16 minutes ago	Up 16 minutes	0.0.0.0:32769->80/tcp	second
efa235511197	stapache	"httpd -D FOREGROUND"	31 minutes ago	Up 31 minutes	0.0.0.0:32768->80/tcp	first



Witaj na stronie stworzonej



Witaj na stronie stworzonej



Witaj na stronie stworzonej

4Z3. Na jednym z poprzednich ćwiczeń instalowany był prywatny rejestr (registry) na podstawie obrazu https://hub.docker.com/_/registry/. Proszę zapoznać się z zawartością Dockerfile dla najnowszej wersji tego repozytorium (i/lub wynikiem działania polecenia `docker history`) Na bazie tego obrazu proszę uruchomić rejestr w taki sposób by:- był on dostępny na porcie 6677,- obrazy były przechowywane w katalogu `~/obrazy_priv` w systemie macierzystym. W sprawozdaniu proszę umieścić dowód na poprawne wykonanie tego zadania.

Mój registry działa na porcie 5000 (nie wiem jak go zmienić, bez mieszania w Dockerfile registry, a żeby przenoszenie „push” działało)

Po kolei

```
~$ mkdir obrazy_priv
```

```
image push registry:latest
```

```
docker run -d --name myregistry --mount  
type=bind,source=/home/student/obrazy_priv,target=/var/lib/r  
egistry -p 5000:5000 registry:latest
```

Później przenoszony jest obraz jak w lab II

```
docker tag redis localgost:5000/redis.local
```

```
docker push localhost:5000/redis.local
```

Sprawdzenie:

```
student@PwCh0-VB:~$ cd obrazy_priv  
student@PwCh0-VB:~/obrazy_priv$ ls  
docker
```

4Z5. Wykorzystując informację z podpunktu E 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 jednego rdzenia. W sprawozdaniu, wykorzystując poznane narzędzia do monitorowania środowiska Docker (tekstowe: `docker inspect ...`, `docker stats ...` jak i graficzne `cadvisor`), należy umieścić zrzuty ekranów, które potwierdzają wprowadzenie w życie wymienionych wyżej ograniczeń.

`docker run -it -d -m 512m --cpuset-cpus 1 --name limiter alpine`

wynik docker stats:

CONTAINER ID	NAME	CPU %	MEM USAGE / LIMIT	MEM %	NET I/O	BLOCK I/O	PIDS
da6a538b3c31	limiter	0.00%	524KiB / 512MiB	0.10%	3.39kB / 0B	0B / 0B	1

student@PwCh0-VB:~/obrazy_priv\$ docker stats limiter

Inspect:

```
"CpuQuota": 0,  
"CpuRealtimePeriod": 0,  
"CpuRealtimeRuntime": 0,  
"CpusetCpus": "1",  
"CpusetMems": "",  
"Devices": []
```

Dla potwierdzenia inspect innego kontenera:

```
student@PwCh0-VB:~/obrazy_priv$ docker inspect myredis |grep Cpu  
"CpuShares": 0,  
"NanoCpus": 0,  
"CpuPeriod": 0,  
"CpuQuota": 0,  
"CpuRealtimePeriod": 0,  
"CpuRealtimeRuntime": 0,  
"CpusetCpus": "",  
"CpusetMems": ""
```