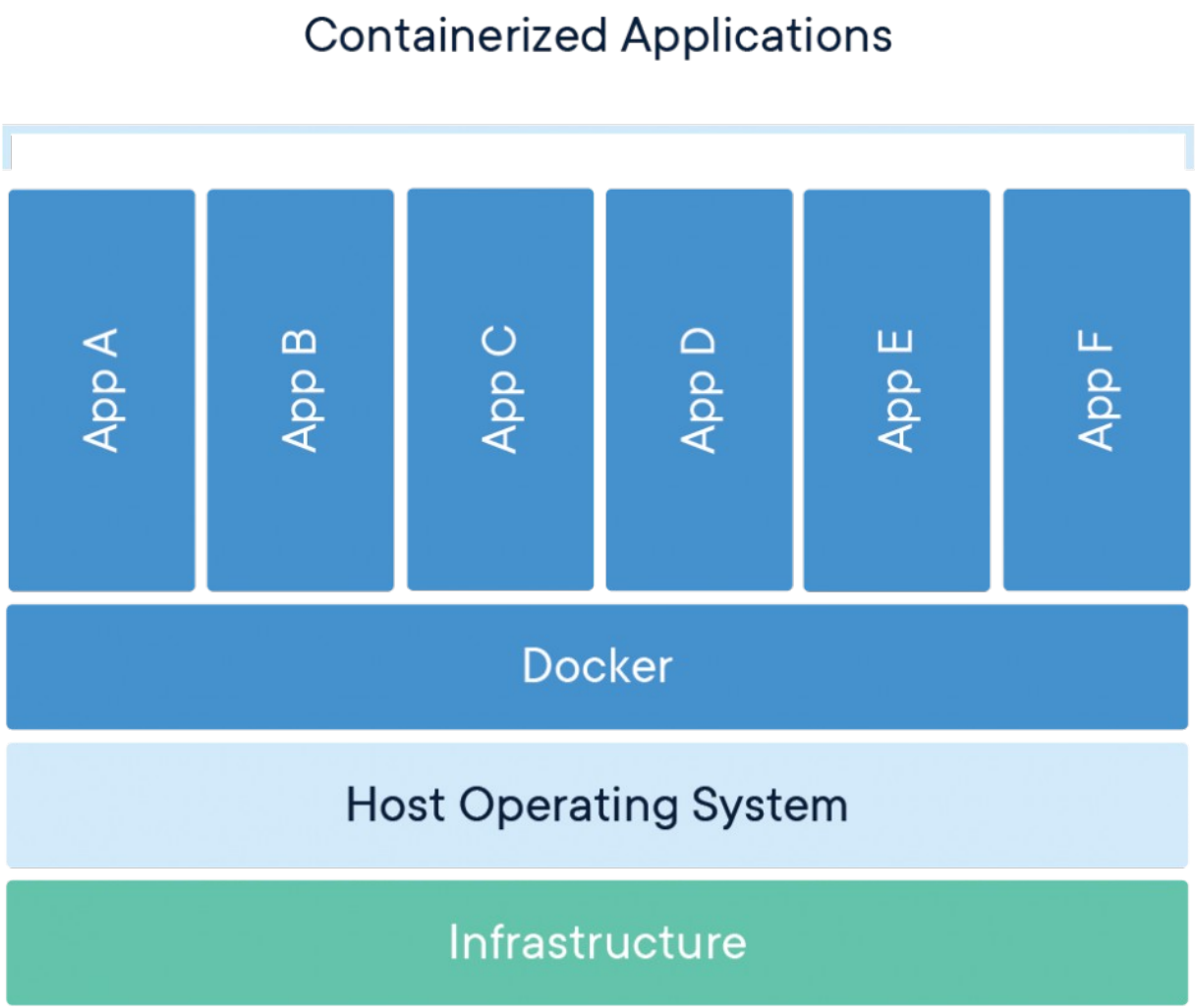
Uniwersytet Wrocławski

Docker

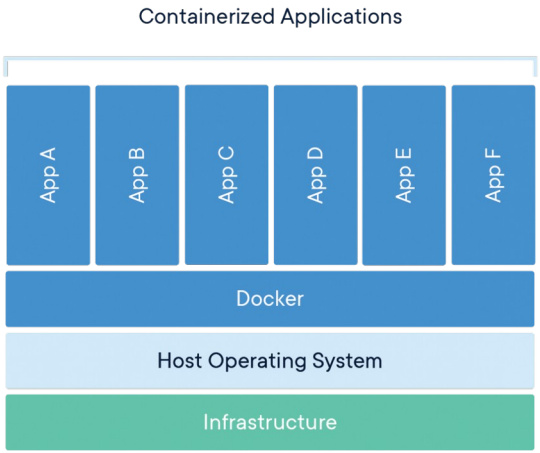
Zbigniew Koza
Wydział Fizyki i Astronomii

Po co są kontenery?



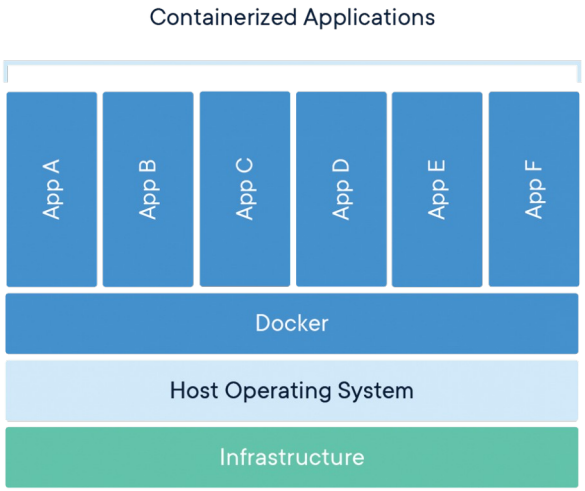
Po co są kontenery?

- Zwykle mamy do czynienia z wieloma komputerami, na każdym inna wersja systemu operacyjnego, bibliotek, aplikacji etc. i ryzyko, że to co działa dziś, działać przestanie po najbliższym uaktualnieniu
- Docker pozwala tanio zbudować standardowe środowiska wykonawcze i łączyć je ze sobą



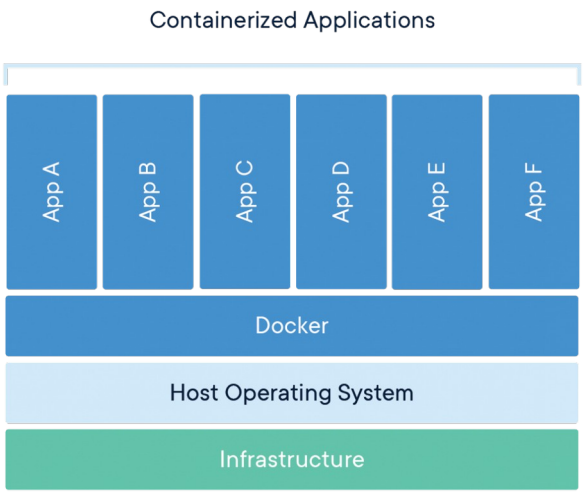
Po co są kontenery?

- Możemy podzielić oprogramowanie (np. komercyjne) na niezależne moduły tworzone w różnych technologiach i mieć pewność, że uda się je uruchomić (na jednej maszynie) i skłonić do współpracy
- Kontenery są odseparowane od siebie, od dockera i warstw niższych



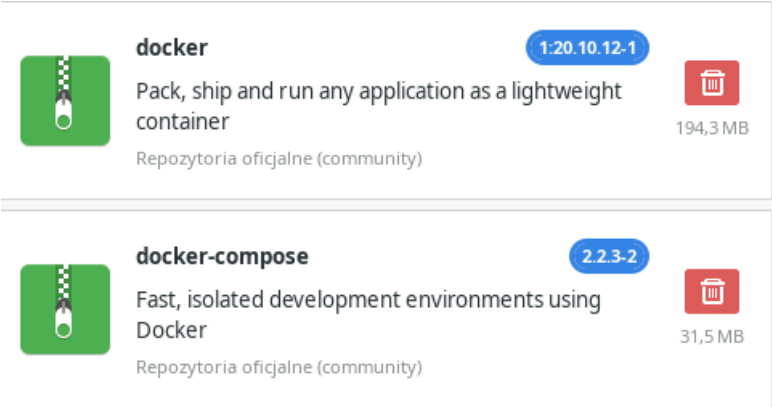
Po co są kontenery?

- frontend,
- backend,
- baz(a/y) danych,
- biblioteki,
- framework(i)
  - Wszystko to można rozwijać, testować, uruchamiać w niezależnych kontenerach na praktycznie każdej maszynie z dowolnym OS w dowolnej konfiguracji



Instalacja, konfiguracja

- Arch Linux:
  - Instalacja pakietów:



  - Konfiguracja:

```
> sudo usermod -a -G docker zkoza
```

```
> groups
sys network power docker lp wheel zkoza
```

```
> sudo systemctl enable docker
> sudo systemctl start docker
> sudo shutdown -r now
```

```
> docker info
Client:
 Context:    default
 Debug Mode: false
 Plugins:
  buildx: Docker Buildx (Docker Inc., v0.7.1-docker)
  compose: Docker Compose (Docker Inc., 2.2.3)

Server:
 Containers: 0
  Running: 0
  Paused: 0
  Stopped: 0
 Images: 0
```

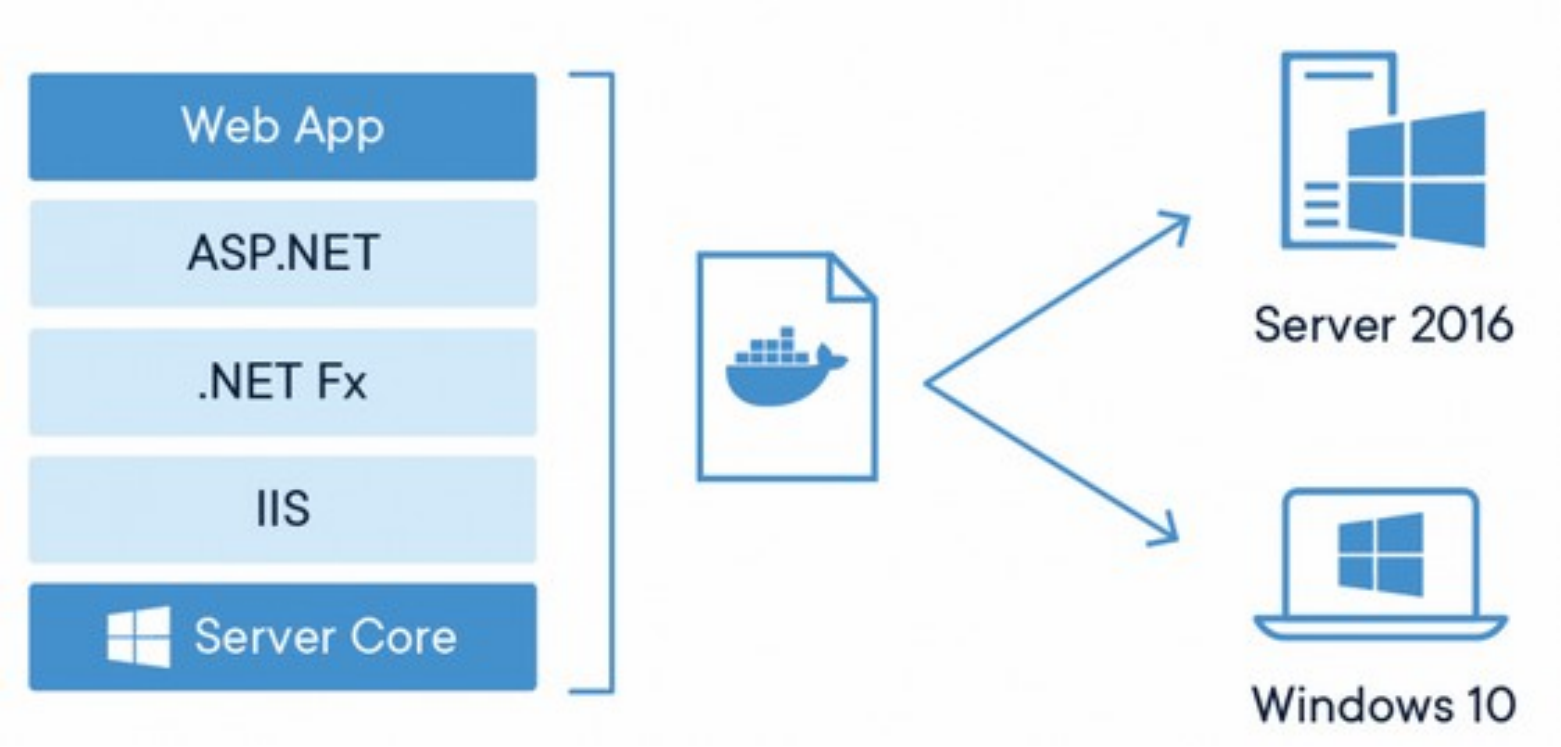
Docker a Linux

- Docker działa pod Linuksem
  - Można uruchomić jednocześnie kilka różnych kontenerów, każdy z inną wersją/dystrybucją Linuksa

```
> docker run -ti ubuntu
root@2a1ff4011c11:/# cat /etc/issue
Ubuntu 20.04.1 LTS \n \l

root@2a1ff4011c11:/# exit
exit
[zkoza@Zbyszek-Dell ~]$ docker run -ti alpine
/ # cat /etc/issue
Welcome to Alpine Linux 3.12
Kernel \r on an \m (\l)

/ # exit
[zkoza@Zbyszek-Dell ~]$ cat /etc/issue
Manjaro Linux \r  (\n) (\l)
```

Docker a (nowy) Windows

- Podobno też działa…



Docker a VMs

- Alternatywą dla kontenerów są maszyny wirtualne
- Docker nie wymaga instalowania osobnego systemu operacyjnego dla każdego kontenera
  - Tworzenie, uruchamianie i „zabijanie” kontenerów jest w dokerze proste, szybkie, naturalne
- Naprawdę chciałbyś na zwykłym laptopie instalować kilka maszyn wirtualnych?

Obraz kontenera

- Obraz (*container image*)
  - Zawiera system plików kontenera i informacje o zmiennych środowiskowych, programie domyślnym etc.
    - Wszystko, co jest potrzebne do uruchamiania programów wewnątrz kontenera
  - Może być przechowywany lokalnie
  - Może być przechowywany zdalnie

docker --help

```
Commands:
  attach      Attach local standard input, output, and
  build       Build an image from a Dockerfile
  commit      Create a new image from a container's cha
  cp          Copy files/folders between a container an
  create      Create a new container
  diff        Inspect changes to files or directories o
  events      Get real time events from the server
  exec        Run a command in a running container
  export      Export a container's filesystem as a tar
  history     Show the history of an image
  images      List images
  import      Import the contents from a tarball to cre
  info        Display system-wide information
  inspect     Return low-level information on Docker ob
  kill        Kill one or more running containers
  load        Load an image from a tar archive or STDIN
  login       Log in to a Docker registry
  logout      Log out from a Docker registry
  logs        Fetch the logs of a container
  pause       Pause all processes within one or more co
  port        List port mappings or a specific mapping
  ps          List containers
  pull        Pull an image or a repository from a regi
  push        Push an image or a repository to a regist
  rename      Rename a container
  restart     Restart one or more containers
  rm          Remove one or more containers
  rmi         Remove one or more images
  run         Run a command in a new container
  save        Save one or more images to a tar archive
  search      Search the Docker Hub for images
  start       Start one or more stopped containers
  stats       Display a live stream of container(s) res
  stop        Stop one or more running containers
  tag         Create a tag TARGET_IMAGE that refers to
  top         Display the running processes of a contai
  unpause     Unpause all processes within one or more
  update      Update configuration of one or more conta
  version     Show the Docker version information
  wait        Block until one or more containers stop,

Run 'docker COMMAND --help' for more information on a c
```

```
Management Commands:
  builder     Manage builds
  buildx*     Docker Buildx (Docker Inc., v
  compose*    Docker Compose (Docker Inc.,
  config      Manage Docker configs
  container   Manage containers
  context     Manage contexts
  image       Manage images
  manifest    Manage Docker image manifests
  network     Manage networks
  node        Manage Swarm nodes
  plugin      Manage plugins
  secret      Manage Docker secrets
  service     Manage services
  stack       Manage Docker stacks
  swarm       Manage Swarm
  system      Manage Docker
  trust       Manage trust on Docker images
```

- *Docker* zawiera własną dokumentację

docker [*command*] --help

```
> docker image --help

Usage:  docker image COMMAND

Manage images

Commands:
  build       Build an image from a Dockerfile
  history     Show the history of an image
  import      Import the contents from a tarball to create a filesystem image
  inspect     Display detailed information on one or more images
  load        Load an image from a tar archive or STDIN
  ls          List images
  prune       Remove unused images
  pull        Pull an image or a repository from a registry
  push        Push an image or a repository to a registry
  rm          Remove one or more images
  save        Save one or more images to a tar archive (streamed to STDOUT by default)
  tag         Create a tag TARGET_IMAGE that refers to SOURCE_IMAGE

Run 'docker image COMMAND --help' for more information on a command.
```

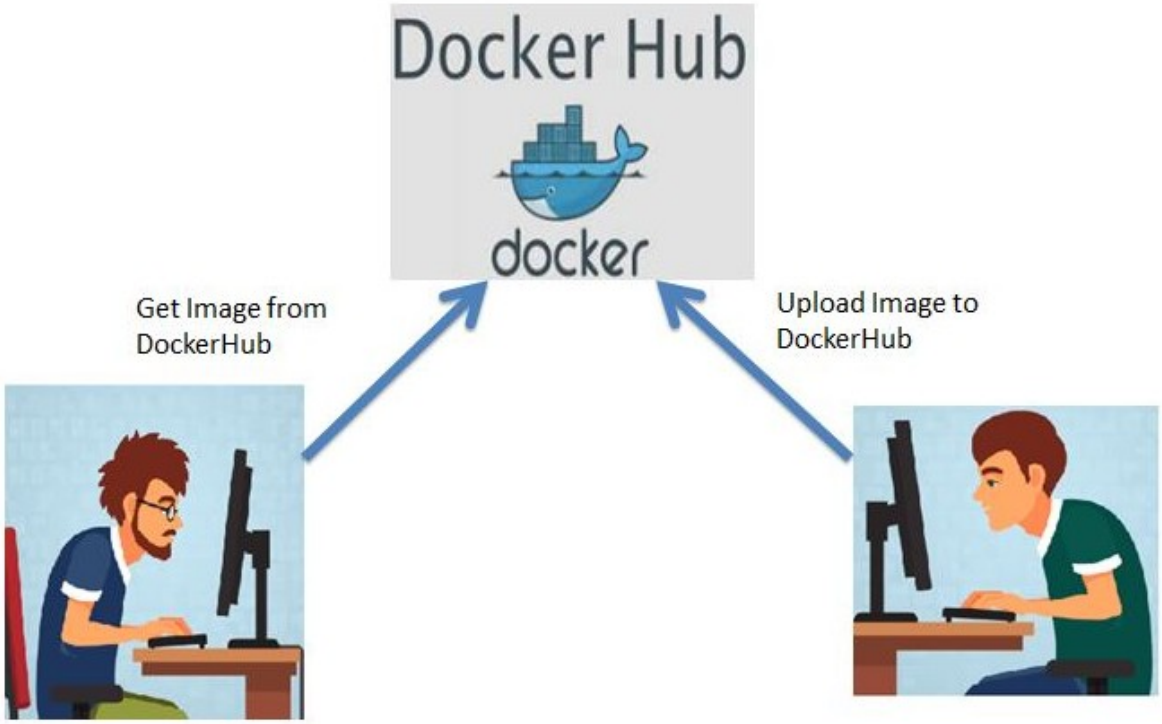
- *Docker* zawiera własną dokumentację

Kilka pytań

- Skąd brać gotowe obrazy?
- Jak z obrazu uruchomić kontener?
- Jak zrządzać obrazami i kontenerami
  - np. które kontenery teraz działają? Jak zakończyć ich działanie? Jak usunąć zbędne obrazy?
- Jak zapisać dane między uruchomieniami tego samego konternera?
- Jak skomunikować kontenery z OS i innymi kontenerami?
- Jak zbudować własny obraz?

Skąd obrazy? Z Docker Hub!

https://hub.docker.com



https://www.javainuse.com/devOps/docker/docker-hub

firebase

Explore Pricing Sign In Sign Up

Community (716)

andreysenov/firebase-tools

w9jds/firebase-action

ridesharing/firebase-push-backend

rambabusaravanan/firebase

Show all 716 hits in Community

Nie musisz mieć konta, żeby móc używać

Filters

Images

Official Images
Official Images Published By Docker

Categories

Analytics

Application Frameworks

Application Infrastructure

Application Services

s for **firebase**. Clear search

Most Popular

eysenov/firebase-tools

By andreysenov • Updated a day ago

500K+ Downloads **13** Stars

Firebase CLI installed globally over the official NodeJS i...

Container Linux x86-64

w9jds/firebase-action

By **w9jds** • Updated 15 days ago

100K+ Downloads **2** Stars

GitHub Action for interacting with Firebase

Container Linux x86-64

Jak pobrać obraz kontenera?

```
> docker run -d -p 80:80 docker/getting-started
Unable to find image 'docker/getting-started:latest' locally
latest: Pulling from docker/getting-started
97518928ae5f: Pull complete
a4e156412037: Pull complete
e0bae2ade5ec: Pull complete
3f3577460f48: Pull complete
e362c27513c3: Pull complete
a2402c2da473: Pull complete
eb65930377cd: Pull complete
69465e074227: Pull complete
Digest: sha256:86093b75a06bf74e3d2125edb77689c8eecf8ed0cb3946573a24a6f71e88cf80
Status: Downloaded newer image for docker/getting-started:latest
c5e83673fd6d2605efa677e717e7173ef49a00f93402cbce3c66aa1b4b4e0311
```

- Po prostu uruchom go tak, jakby już był pobrany
- Docker przechowuje pobrane obrazy w specjalnym katalogu

docker run

- `> docker run`

```
> docker help run

Usage:  docker run [OPTIONS] IMAGE [COMMAND] [ARG...]

Run a command in a new container
```

- Wybrane opcje:

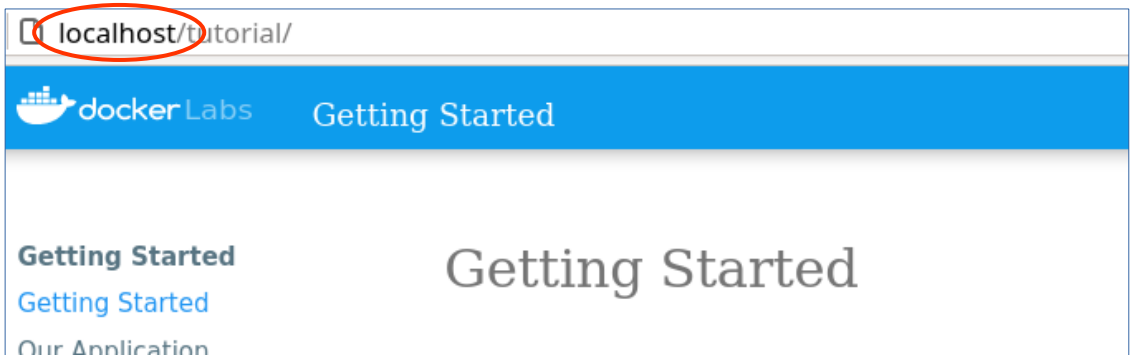
częste pary

`-a`	Attach to STDIN, STDOUT or STDERR
`-c,`	CPU shares (relative weight)
`-d,`	**Run container in background**
`-e`	Set environment variables
`-h`	Container host name
`-i`	**Keep STDIN open even if not attached**
`-m`	Memory limit
`-p`	**Publish a container's port(s) to the host**
`-v`	**Bind mount a volume**
`-w`	Working directory inside the container
`-t`	**Allocate a pseudo-TTY**

docker run -dp

```
> docker run -dp 80:80 docker/getting-started
```

- Uruchom obraz
  `docker/getting-started`
  w tle (-d) i skojarz port 80 komputera (*host OS*) z portem 80 obrazu
  - Proces „w tle” nie ma terminala
  - Port 80 do domyślny port protokołu HTTP



docker run -ti

```
> docker run -ti ubuntu
root@a97c93338c81:/# cat /etc/issue
Ubuntu 20.04.1 LTS \n \l

root@a97c93338c81:/# du -hd1
20K     ./run
4.0K    ./opt
4.0K    ./media
0       ./proc
624K    ./etc
4.0K    ./srv
12K     ./root
73M     ./usr
4.0K    ./tmp
4.5M    ./var
4.0K    ./mnt
0       ./dev
4.0K    ./boot
4.0K    ./home
0       ./sys
78M     .
```

- `-t` = „przypisz terminal”
- `-i` = „uruchom jako proces interaktywny”

Plik Dockerfile

- **Dockerfile** to skrytpt używany do budowania własnego obrazu kontenera

```
FROM node:alpine
WORKDIR /app
COPY . .
RUN yarn install --production
RUN yarn add express
RUN yarn add sqlite3
CMD ["node", "app/src/index.js"]
```

- Zaczynamy od obrazu „standardowego” `node:alpine`
- Ustalamy katalog roboczy
- Kopiujemy pliki obrazu do kontenera
- Instalujemy pakiety `express` oraz `sqlite3` (metodą z Alpine Linux)
- Ustalamy polecenie domyślnie wykonywane na kontenerze

Dockerfile

- **`Dockerfile`** to skrytpt używany do budowania własnego obrazu kontenera

```
FROM node:12-alpine
RUN apk add --no-cache python2 g++ make
COPY . .
WORKDIR /app
RUN yarn install --production
CMD ["node", "src/index.js"]
```

Alpine Linux jest popularną w Dokerze, minimalistyczną dystrybucją Linuksa

node.js

- Zaczynamy od obrazu „standardowego” `node:alpine`
- Instalujemy python2, g++, make
- Kopiujemy pliki z katalogu bieżącego komputera do kontenera
- Ustalamy katalog roboczy
- Instalujemy aplikację (NodeJs)
- Ustalamy polecenie domyślnie wykonywane na kontenerze

COPY . .

- Pliki lokalne:

```
> ls -l
razem 8
drwxr-xr-x 4 zkoza zkoza 4096 01-29 01:07 app
-rw-r--r-- 1 zkoza zkoza  223 01-29 16:02 Dockerfile
```

```
> ls -l app
razem 172
-rw-r--r-- 1 zkoza zkoza    646 01-29 01:07 package.json
drwxr-xr-x 4 zkoza zkoza   4096 01-29 01:07 spec
drwxr-xr-x 5 zkoza zkoza   4096 01-29 01:07 src
-rw-r--r-- 1 zkoza zkoza 162208 01-29 01:07 yarn.lock
```

- Pliki w kontenerze:

Prog. do uruchomienia

```
> docker run -ti getting-started-2 ls -l /
total 76
-rw-r--r--    1 root     root           223 Jan 29 15:02 Dockerfile
drwxr-xr-x    1 root     root          4096 Jan 29 15:02 app
drwxr-xr-x    1 root     root          4096 Jan 12 00:13 bin
drwxr-xr-x    5 root     root           360 Jan 29 15:16 dev
drwxr-xr-x    1 root     root          4096 Jan 29 15:15 etc
drwxr-xr-x    1 root     root          4096 Jan 11 21:23 home
drwxr-xr-x    1 root     root          4096 Jan 12 00:13 lib
drwxr-xr-x    5 root     root          4096 Nov 24 09:20 media
drwxr-xr-x    2 root     root          4096 Nov 24 09:20 mnt
drwxr-xr-x    1 root     root          4096 Jan 12 00:13 opt
dr-xr-xr-x  344 root     root             0 Jan 29 15:16 proc
drwx------    1 root     root          4096 Jan 12 00:13 root
drwxr-xr-x    2 root     root          4096 Nov 24 09:20 run
drwxr-xr-x    2 root     root          4096 Nov 24 09:20 sbin
drwxr-xr-x    2 root     root          4096 Nov 24 09:20 srv
dr-xr-xr-x   13 root     root             0 Jan 29 15:16 sys
drwxrwxrwt    1 root     root          4096 Jan 29 15:02 tmp
drwxr-xr-x    1 root     root          4096 Jan 28 23:42 usr
drwxr-xr-x    1 root     root          4096 Nov 24 09:20 var
```

```
> docker run -ti getting-started-2 ls -l /app
total 188
drwxr-xr-x  163 root     root          4096 Jan 29 15:02 node_modules
-rw-r--r--    1 root     root           646 Jan 29 00:07 package.json
drwxr-xr-x    4 root     root          4096 Jan 29 00:07 spec
drwxr-xr-x    5 root     root          4096 Jan 29 00:07 src
-rw-r--r--    1 root     root        174600 Jan 29 15:02 yarn.lock
```

- Dockerfile:

```
> cat Dockerfile
FROM node:12-alpine
RUN apk add --no-cache python2 g++ make
COPY . .
WORKDIR /app
RUN yarn install --production
CMD ["node", "src/index.js"]
```

apk = Alpine package keeper; yarn = software packaging system for Node.js

Budowanie obrazu: `docker build`

- Dockerfile:

```
FROM node:12-alpine
RUN apk add --no-cache python2 g++ make
COPY . .
WORKDIR /app
RUN yarn install --production
CMD ["node", "src/index.js"]
```

> docker build -t getting-started-node .

- -t = „tag” (nazwa czytelna dla człowieka)

Gdzie jest Dockerfie

```
FROM node
WORKDIR /app
COPY . .
RUN yarn install --production
RUN yarn add express
RUN yarn add sqlite3
CMD ["node", "app/src/index.js"]
```

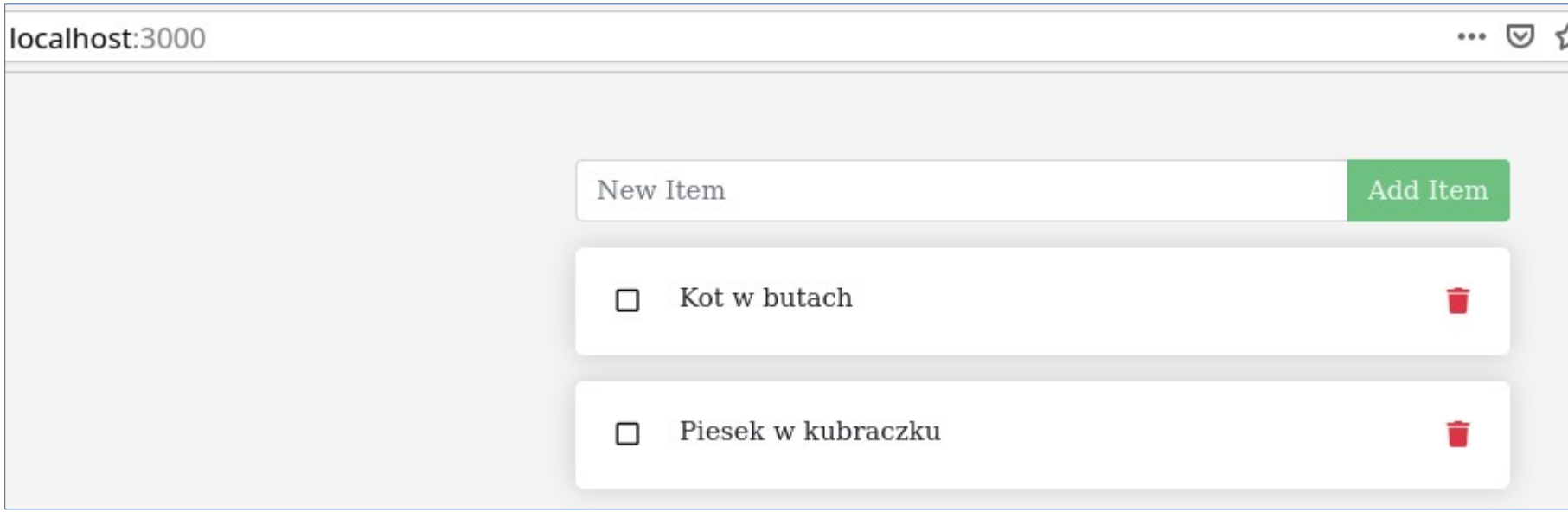
```
> docker build -t getting-started-node .
Sending build context to Docker daemon   6.42MB
Step 1/7 : FROM node
 ---> ea27efc47a35
Step 2/7 : WORKDIR /app
 ---> Using cache
 ---> 1fb91bdf7b7d
Step 3/7 : COPY . .
 ---> Using cache
 ---> c09f43461f22
Step 4/7 : RUN yarn install --production
 ---> Using cache
 ---> 29adcaa278a9
Step 5/7 : RUN yarn add express
 ---> Using cache
 ---> 16c8239f6501
Step 6/7 : RUN yarn add sqlite3
 ---> Using cache
 ---> 426f3eb57ca3
Step 7/7 : CMD ["node", "app/src/index.js"]
 ---> Using cache
 ---> 32a11ebe1116
Successfully built 32a11ebe1116
Successfully tagged getting-started-node:latest
```

- Budowanie przebiega warstwowo, każda komenda w `Dockerfile` to nowy obraz ea27efc47a35 → 1fb91bdf7b7d → … 32a11ebe1116

Uruchamianie

- > docker run

```
> docker run -dp 3000:3000 getting-started-node
```



Lista działających kontenerów

```
> docker ps
```

```
> docker ps
CONTAINER ID   IMAGE                    COMMAND                  CREATED         STATUS         PORTS                    NAMES
9292dcb4c69a   getting-started-node     "docker-entrypoint.s…"   2 minutes ago   Up 2 minutes   0.0.0.0:3000->3000/tcp   boring_dir
df01614b2160   docker/getting-started   "/docker-entrypoint.…"   6 hours ago     Up 6 hours     0.0.0.0:80->80/tcp       charming_k
```

Zamykanie kontenera

```
> docker ps
> docker stop container_id
> docker rm container_id
```

```
> docker ps
CONTAINER ID   IMAGE
9292dcb4c69a   getting-started-node
df01614b2160   docker/getting-started
```

```
> docker stop 9292dcb4c69a
9292dcb4c69a
```

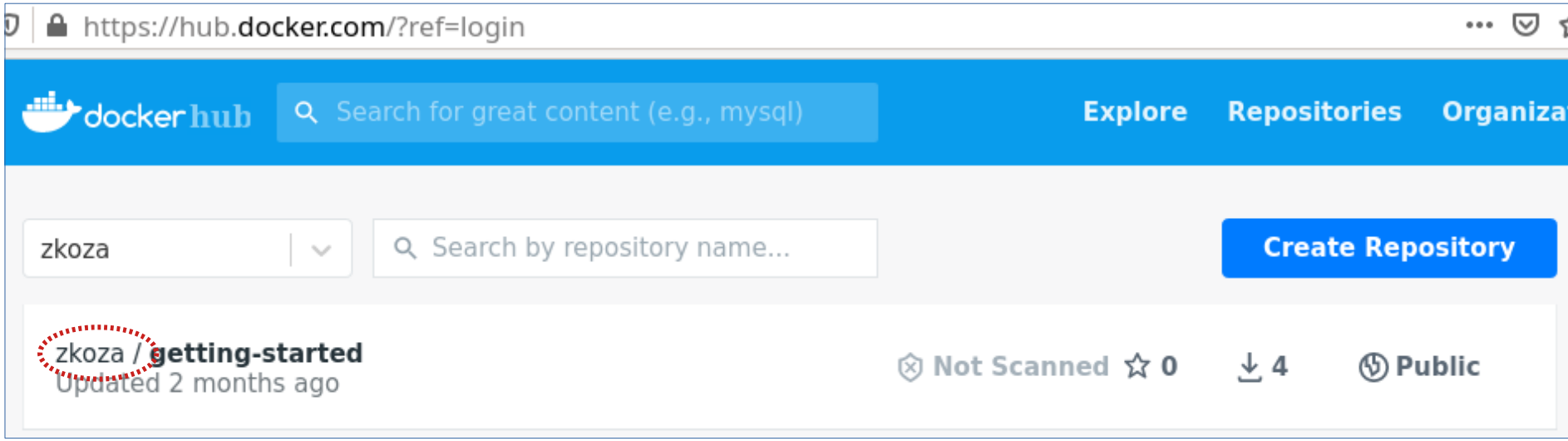
```
> docker rm 9292dcb4c69a
9292dcb4c69a
```

docker push

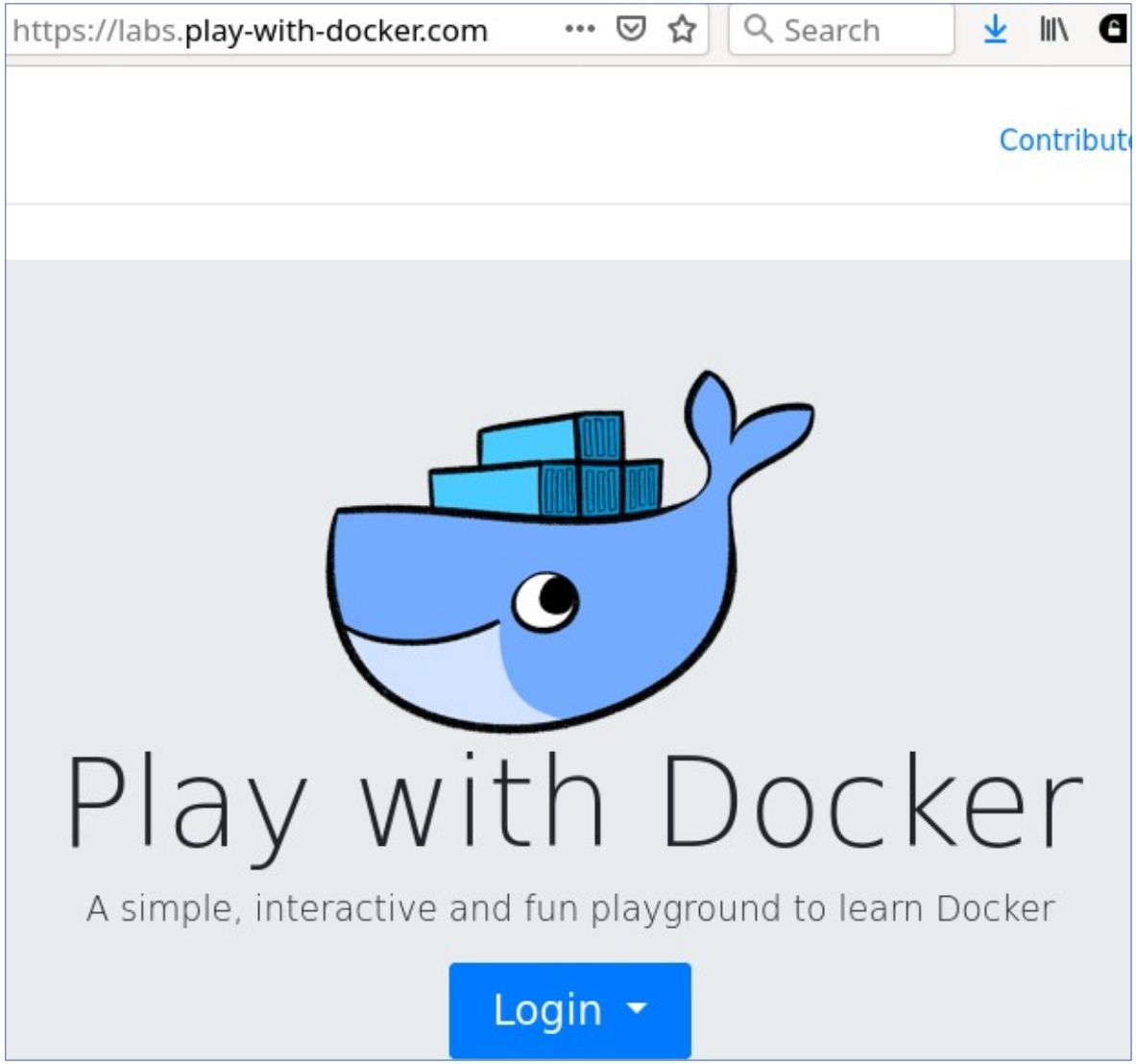
```
> docker login -u zkoza
> docker tag getting-started-node zkoza/getting-started-node
> docker push zkoza/getting-started-node
```

```
Using default tag: latest
The push refers to repository [docker.io/zkoza/getting-started-node]
71ef7e7d55ea: Pushing [==================================================>]  43.02MB/45.39MB
9051ef454004: Pushed
903060731581: Pushing [==========================>]  52.89MB/104.2MB
e30959bf4db1: Pushed
30fed8b901aa: Pushed
6f903a63aec0: Pushed
9459233b6a63: Pushed
7ed9e3d1c5f1: Pushing [====================>]  36.74MB/92.95MB
fdba6a5d9dd7: Pushed
07700abd910e: Pushing [===>]  37.95MB/561.3MB
edfb8ee7c346: Pushing [====>]  13.12MB/141.8MB
aa817488a0dd: Waiting
74825a980b6d: Waiting
1fb0a31fe7c2: Preparing
```

docker push


https://hub.docker.com/?ref=login
docker hub
Search for great content (e.g., mysql)
Explore
Repositories
Organiza
zkoza
Search by repository name...
Create Repository
zkoza / getting-started
Updated 2 months ago
Not Scanned
0
4
Public

Play with docker



Play with docker

03:58:30

CLOSE SESSION

Instances bu se

+ ADD NEW INSTANCE

192.168.0.28
node1

c0c8s9k3_c0c8sfs34gag0085qktg

IP
192.168.0.28

OPEN PORT

Memory
1.06% (42.23MiB / 3.906GiB)

CPU
1.07%

SSH
ssh ip172-18-0-29-c0c8s9k34gag0085qkt0@direct.labs.play-with-d content_copy

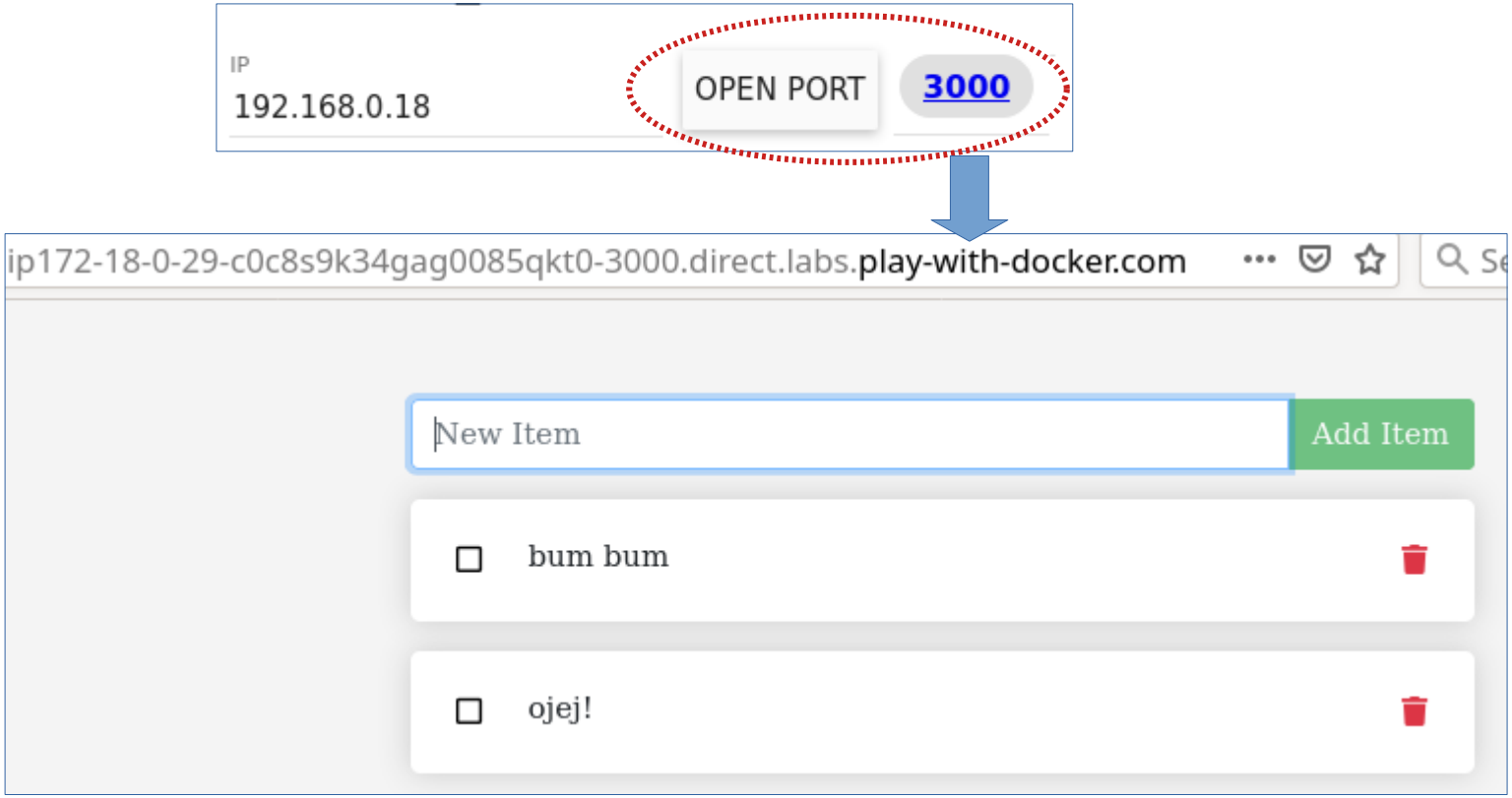
DELETE

EDITOR

```
##################################################################
#                          WARNING!!!!                           #
# This is a sandbox environment. Using personal credentials      #
# is HIGHLY! discouraged. Any consequences of doing so are       #
# completely the user's responsibilites.                         #
#                                                                #
# The PWD team.                                                  #
##################################################################
[node1] (local) root@192.168.0.28 ~
$
```

Play with docker

```
[node1] (local) root@192.168.0.18 ~
$ docker run -dp 3000:3000 zkoza/getting-started
```



Play with docker

```
[zkoza@Zbyszek-Dell app]$ ssh ip172-18-0-29-c0c8s9k34gag0085qkt0@direct.labs.play-with-docker.com
The authenticity of host 'direct.labs.play-with-docker.com (40.76.55.146)' can't be established.
RSA key fingerprint is SHA256:UyqFRi42lglohSOPKn6Hh9M83Y5Ic9IQn1PTHYqOjEA.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'direct.labs.play-with-docker.com,40.76.55.146' (RSA) to the list of known hosts.
Connecting to 52.170.84.189:8022
#################################################################
#                          WARNING!!!!                          #
# This is a sandbox environment. Using personal credentials     #
# is HIGHLY! discouraged. Any consequences of doing so are      #
# completely the user's responsibilites.                        #
#                                                               #
# The PWD team.                                                 #
#################################################################
[node1] (local) root@192.168.0.28 ~
$ cat /etc/issue
Welcome to Alpine Linux 3.12
Kernel \r on an \m (\l)
```

docker exe

konsola nr 1:

```
> docker run  -ti ubuntu
root@821fe8578aa5:/# echo "ala" > ola
```

konsola nr 2:

```
> docker ps
CONTAINER ID   IMAGE                    COMMAND                  CREATED
821fe8578aa5   ubuntu                   "/bin/bash"              About a minut
df01614b2160   docker/getting-started   "/docker-entrypoint.…"   11 hours ago
[zkoza@Zbyszek-Dell app]$ docker exec 821fe8578aa5 cat ola
ala
```

Trwałość danych

- Po zamknięciu kontenera wszelkie zapisane w nim dane nikną...

Volumeny dockera

```
> docker volume create todo-db
```

Nazwa wolumenu

- Możliwe opcje:
  - `create` Utwórz wolumen
  - `inspect` Wyświetl informacje o wolumenie
  - `ls` Wyświetl listę wolumenów
  - `prune` Usuń wszystkie nieużywane lokalne wolumeny
  - `rm` Usuń wolumen

- Są trwale przechowywane na dysku

Wolumeny dokera

```
> docker run -dp 3000:3000 -v todo-db:/etc/todos getting-started-node
```

```
> docker run -ti -v todo-db:/etc/todos getting-started-node bash
root@db9448d55fa8:/app# ls -l /etc/todos/
total 8
-rw-r--r-- 1 root root 8192 Feb  2 00:10 todo.db
```

SQLite database

- Wszystko, co zapiszemy w katalogu `/etc/todos`, będzie tam wciąż dostępne w każdej kolejnej sesji

Praca z wolumenami

```
> docker volume ls
DRIVER    VOLUME NAME
local     todo-db
```

```
> docker volume inspect todo-db
[
    {
        "CreatedAt": "2022-01-29T16:54:35+01:00",
        "Driver": "local",
        "Labels": null,
        "Mountpoint": "/var/lib/docker/volumes/todo-db/_data",
        "Name": "todo-db",
        "Options": null,
        "Scope": "local"
    }
]
```

```
> sudo file /var/lib/docker/volumes/todo-db/_data/todo.db
/var/lib/docker/volumes/todo-db/_data/todo.db: SQLite 3.x database, last writt
en using SQLite version 3034000, file counter 1, database pages 2, cookie 0x1,
 schema 4, UTF-8, version-valid-for 1
```

Bind mounts

```
docker run -dp 3000:3000  \
-w /app \
-v "$(pwd):/app" node \
bash -c "yarn install && yarn run dev"
```

- `-w :` ustala katalog roboczy
- `-v :` definiuje „bind mount”, czyli montuje fizyczny katalog `$(pwd)` jako katalog `/app` kontenera

Bind mounts

- Umożliwiają podmontowanie (podpięcie) katalogu z dysku (lokalnego, sieciowego) do katalogu dokera
  - Np. piszesz „u siebie”, testujesz w kontenerze
  - Możesz zmienić kod źródłowy strony www wyświetlanej w kontenerze i po prostu ją odświeżyć
  - Ta sama składnia, jak w przypadku wolumenów, tylko źródło jest podawane jako „absolute path”

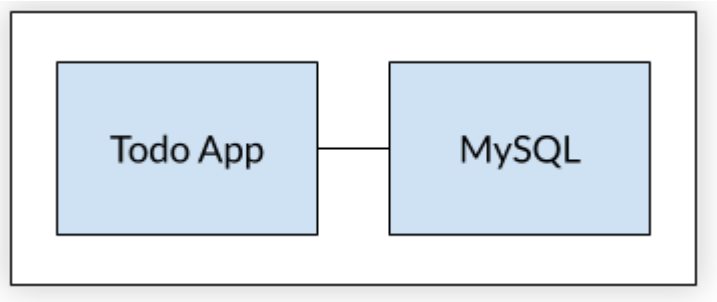
docker logs

- Co się dzieje z moim kontenerem?

```
> docker logs -f 81848e226eebc89e65489a8dad28f39d6a8f06cd79
yarn install v1.22.5
[1/4] Resolving packages...
[2/4] Fetching packages...
info fsevents@1.2.9: The platform "linux" is incompatible w
info "fsevents@1.2.9" is an optional dependency and failed
[3/4] Linking dependencies...
[4/4] Building fresh packages...
Done in 97.19s.
yarn run v1.22.5
$ nodemon src/index.js
[nodemon] 1.19.2
[nodemon] to restart at any time, enter `rs`
[nodemon] watching dir(s): *.*
[nodemon] starting `node src/index.js`
Using sqlite database at /etc/todos/todo.db
Listening on port 3000
```

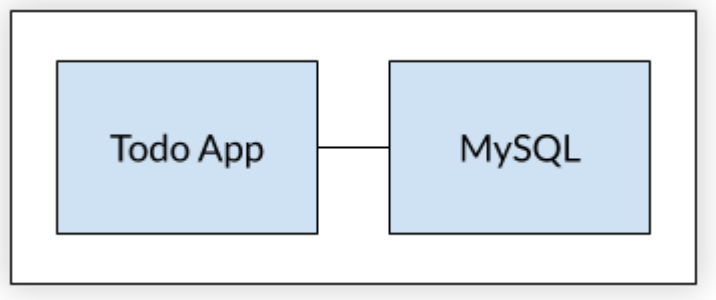
Łączenie kontenerów

- **Kontenerów może być dużo**
- Każdy powinien odpowiadać za jedną funkcjonalność, ale swoją funkcję niech wykonuje naprawdę dobrze



container networking

```
> docker network create todo-app
```



- Tworzy „sieć lokalną” dla kontenerów
- Kontenery podpięte do tej samej sieci mogą się ze sobą komunikować

MySQL

```
docker run -d \
 --network todo-app --network-alias mysql \
 -v todo-mysql-data:/var/lib/mysql \
 -e MYSQL_ROOT_PASSWORD=secret \
 -e MYSQL_DATABASE=todos \
 mysql:5.7
```

Czy to zadziałało?

```
> docker exec -it 50e846051fa09b26d32554b065ed9da mysql -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 3
Server version: 5.7.33 MySQL Community Server (GPL)

Copyright (c) 2000, 2021, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
| todos              |
+--------------------+
5 rows in set (0.00 sec)

mysql>
```

Jak podłączyć kontener do sieci?

```
> docker run -it --network todo-app
    nicolaka/netshoot
```

Wcześniej utworzona sieć

Ten kontener zawiera liczne narzędzia diagnostyczne sieci

```
Welcome to Netshoot! (github.com/nicolaka/netshoot)
root @ /
 [1] 🐳  → dig mysql

; <<>> DiG 9.14.12 <<>> mysql
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 28410
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;mysql.                         IN      A

;; ANSWER SECTION:
mysql.                  600     IN      A       172.21.0.2

;; Query time: 0 msec
;; SERVER: 127.0.0.11#53(127.0.0.11)
;; WHEN: Tue Feb 02 01:04:59 UTC 2021
;; MSG SIZE  rcvd: 44


root @ /
 [2] 🐳  → ping mysql
PING mysql (172.21.0.2) 56(84) bytes of data.
64 bytes from mysql.todo-app (172.21.0.2): icmp_seq=1 ttl=64 time=0.492 ms
64 bytes from mysql.todo-app (172.21.0.2): icmp_seq=2 ttl=64 time=0.168 ms
64 bytes from mysql.todo-app (172.21.0.2): icmp_seq=3 ttl=64 time=0.167 ms
64 bytes from mysql.todo-app (172.21.0.2): icmp_seq=4 ttl=64 time=0.159 ms
^C
--- mysql ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3037ms
rtt min/avg/max/mdev = 0.159/0.246/0.492/0.141 ms
```

--network-alias

- Kontener podłączony do sieci (--network) rozpoznaje jej nazwę jako alias adresu sieciowego

Środowisko (-e)

```
docker run -dp 3000:3000 \
  -w /app -v "$(pwd):/app" \
  --network todo-app \
  -e MYSQL_HOST=mysql \
  -e MYSQL_USER=root \
  -e MYSQL_PASSWORD=secret \
  -e MYSQL_DB=todos \
  node:12-alpine \
  sh -c "yarn install && yarn run dev"
```

```
if (process.env.MYSQL_HOST) module.exports = require('./mysql');
else module.exports = require('./sqlite');
```

1

```
const {
    MYSQL_HOST: HOST,
    MYSQL_HOST_FILE: HOST_FILE,
    MYSQL_USER: USER,
    MYSQL_USER_FILE: USER_FILE,
    MYSQL_PASSWORD: PASSWORD,
    MYSQL_PASSWORD_FILE: PASSWORD_FILE,
    MYSQL_DB: DB,
    MYSQL_DB_FILE: DB_FILE,
} = process.env;
```

2

Fragmenty kodu aplikacji NodeJS

4

```
pool = mysql.createPool({
    connectionLimit: 5,
    host,
    user,
    password,
    database,
});
```

```
const host = HOST_FILE ? fs.readFileSync(HOST_FILE) : HOST;
```

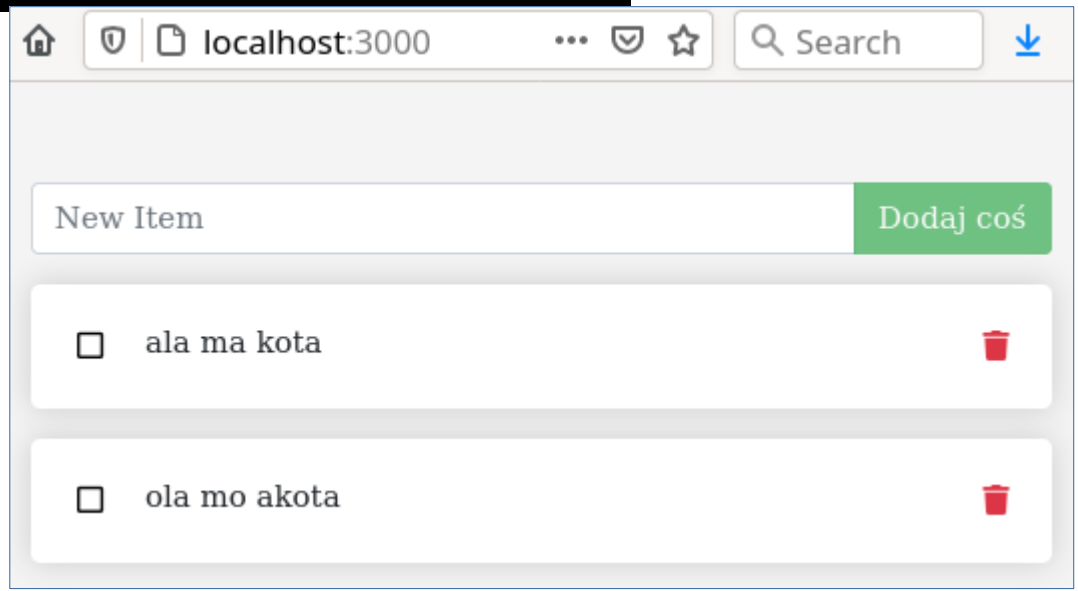
3

Sprawdzamy...

```
> docker ps
CONTAINER ID   IMAGE                    COMMAND
084a05c35349   node:12-alpine           "docker-entrypo
50e846051fa0   mysql:5.7                "docker-entrypo
df01614b2160   docker/getting-started   "/docker-entryp
```

```
docker exec -it 50e8460 mysql -p todos
```

```
mysql> select * from todo_items;
+--------------------------------------+--------------+-----------+
| id                                   | name         | completed |
+--------------------------------------+--------------+-----------+
| c28cdd12-72f2-4e88-8b09-8d33beedc976 | ala ma kota  |         0 |
| 4cbbcc67-13b7-47f3-91bc-328dc4d6b46f | ola mo akota |         0 |
+--------------------------------------+--------------+-----------+
```

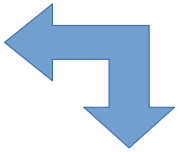


`docker-compose`

- Ręczne zarządzanie wieloma kontenerami jest żmudne i podatne na błędy
- Rozwiązanie: **skrypt YAML**
  - Nazwa pliku: **`docker-compose.yml`**
  - Coś na podobieństwo skryptów GitHub-a etc.
- Automatyzuje, na dowolnej platformie, uruchamianie (wielu) kontenerów (np. lokalne testy albo zarządzanie aplikacją w chmurze)

```
docker run -dp 3000:3000 \
  -w /app -v "$(pwd):/app" \
  --network todo-app \
  -e MYSQL_HOST=mysql \
  -e MYSQL_USER=root \
  -e MYSQL_PASSWORD=secret \
  -e MYSQL_DB=todos \
  node:12-alpine \
  sh -c "yarn install && yarn run dev"
```

docker-compose.yml



Alias sieciowy
Obraz kontenera
Polecenie
Skojarzone porty
Katalog roboczy
Wolumeny
Środowisko

```yaml
version: "3.7"

services:
  app:
    image: node:12-alpine
    command: sh -c "yarn install && yarn run dev"
    ports:
      - 3000:3000
    working_dir: /app
    volumes:
      - ./:/app
    environment:
      MYSQL_HOST: mysql
      MYSQL_USER: root
      MYSQL_PASSWORD: secret
      MYSQL_DB: todos
```

```
docker run -d \
  --network todo-app --network-alias mysql \
  -v todo-mysql-data:/var/lib/mysql \
  -e MYSQL_ROOT_PASSWORD=secret \
  -e MYSQL_DATABASE=todos \
  mysql:5.7
```

docker-compose.yml

MySQL:

```yaml
version: "3.7"

services:
  app:
    # The app service definition
  mysql:
    image: mysql:5.7
    volumes:
      - todo-mysql-data:/var/lib/mysql
    environment:
      MYSQL_ROOT_PASSWORD: secret
      MYSQL_DATABASE: todos

volumes:
  todo-mysql-data:
```

Tworzone wolumeny

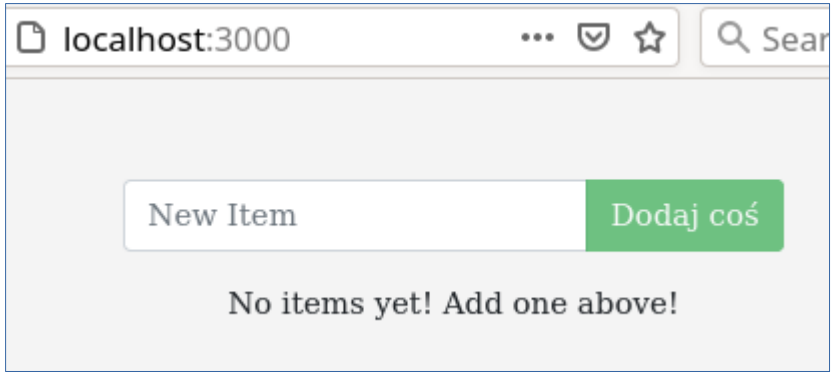
docker-compose.yml

```yaml
version: "3.7"

services:
  app:
    image: node:12-alpine
    command: sh -c "yarn install && yarn run dev"
    ports:
      - 3000:3000
    working_dir: /app
    volumes:
      - ./:/app
    environment:
      MYSQL_HOST: mysql
      MYSQL_USER: root
      MYSQL_PASSWORD: secret
      MYSQL_DB: todos

  mysql:
    image: mysql:5.7
    volumes:
      - todo-mysql-data:/var/lib/mysql
    environment:
      MYSQL_ROOT_PASSWORD: secret
      MYSQL_DATABASE: todos

volumes:
  todo-mysql-data:
```

Docker-compose: uruchomienie kontenerów

- Docker-compose up -d

```
> docker-compose up -d
Creating app_mysql_1 ... done
Creating app_app_1   ... done
[zkoza@Zbyszek-Dell app]$ docker ps
CONTAINER ID   IMAGE                  COMMAND                  CREATED          STATUS          PORTS                    NAMES
69656aa5f677   node:12-alpine         "docker-entrypoint.s…"   14 seconds ago   Up 13 seconds   0.0.0.0:3000->3000/tcp   app_app_1
bce51e995b38   mysql:5.7              "docker-entrypoint.s…"   14 seconds ago   Up 13 seconds   3306/tcp, 33060/tcp      app_mysql_1
df01614b2160   docker/getting-started "/docker-entrypoint.…"   21 hours ago     Up 21 hours     0.0.0.0:80->80/tcp       charming_knuth
```



Literatura, inspiracje:

- https://www.docker.com/101-tutorial
- http://dast.webd.pl/podstawy-dockera/
- https://www.geeksforgeeks.org/docker-compose/