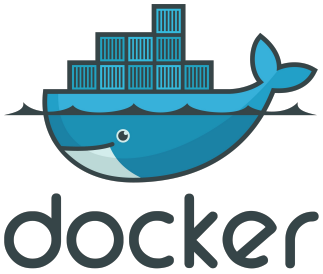


Podstawy oraz praktyczne zastosowanie Dockera

Marcin Dargacz

InfoShare Academy 30 listopada 2018 r.

Moja prezentacja

https://github.com/infoshareacademy/jjdd5-materialy-docker

Folder: ***prezentacja***

BLUE MEDIA

AitonCaldwell

POLSKA PRESS GRUPA

infoShare academy

Poznajmy się!

1. Imię
2. Oczekiwania względem warsztatu.

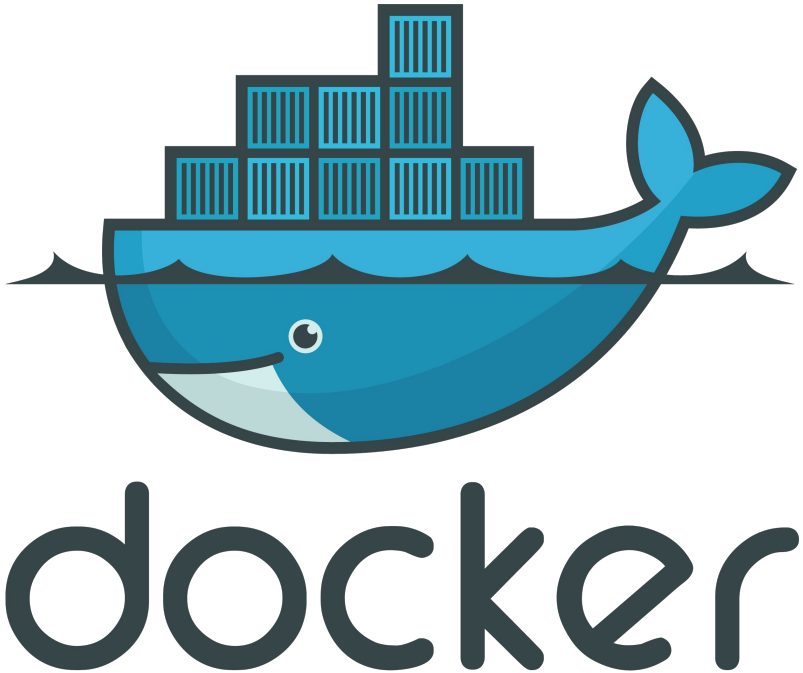
źródło: https://pixabay.com

Zasady

- mówimy sobie po imieniu
- zadajemy pytania na bieżąco
- wszyscy na tym samym etapie - w razie potrzeby pomagamy sobie wzajemnie

Cele warsztatu

- poznanie i zrozumienie możliwości narzędzi **docker** i **docker compose**
- umiejętność budowania prostej architektury systemu aplikacji JAVA z wykorzystaniem **dockera**



źródło: https://docker.com

Agenda

1. Wstęp teoretyczny
2. Podstawy narzędzia **docker**
3. Budowanie własnych **obrazów**
4. Uruchamianie **kontenerów**
5. Budowanie własnej architektury aplikacji w JAVA z wykorzystaniem narzędzia **docker-compose**
6. * Przegląd ciekawych narzędzi
7. * Inne :)

infoShare academy

Na wstępie ...

Wersja narzędzia

```
#C docker -v
```

- w taki sposób można łatwo sprawdzić, czy docker jest zainstalowany na naszym komputerze

Co zrobić gdy masz za starą wersję?

```
#C docker run -it docker
```

- Można uruchomić dockera w dockerze 

Szczegółowe informacje na temat narzędzia

#C docker info

- docker wymaga **64 bitowej** wersji systemu
- wykorzystuje możliwości jądra systemu Linux

Pierwszy raz

2013 r. - wystąpienie Solomona Hykes na konferencji PyCon

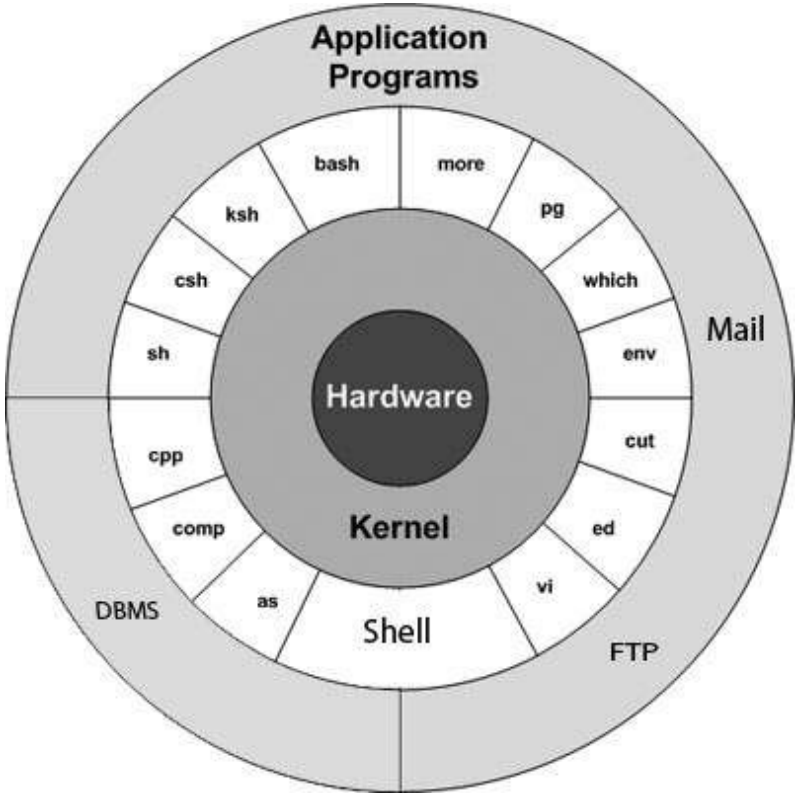
#R https://youtu.be/wW9CAH9nSLs

dotCloud

Zwykła wirtualizacja **vs** Docker

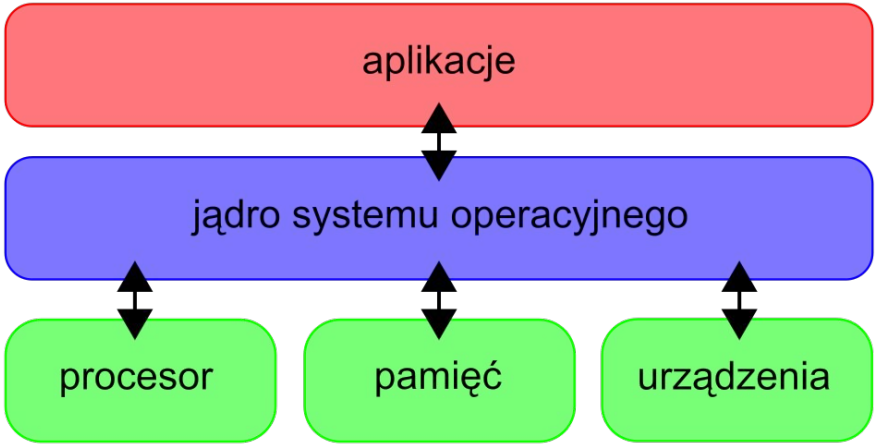
źródło: https://pixabay.com/

Ale zanim ...
Jak zbudowany jest System operacyjny?



Źródło: https://www.tutorialspoint.com/unix/unix-getting-started.htm

Jaka jest rola jądra?

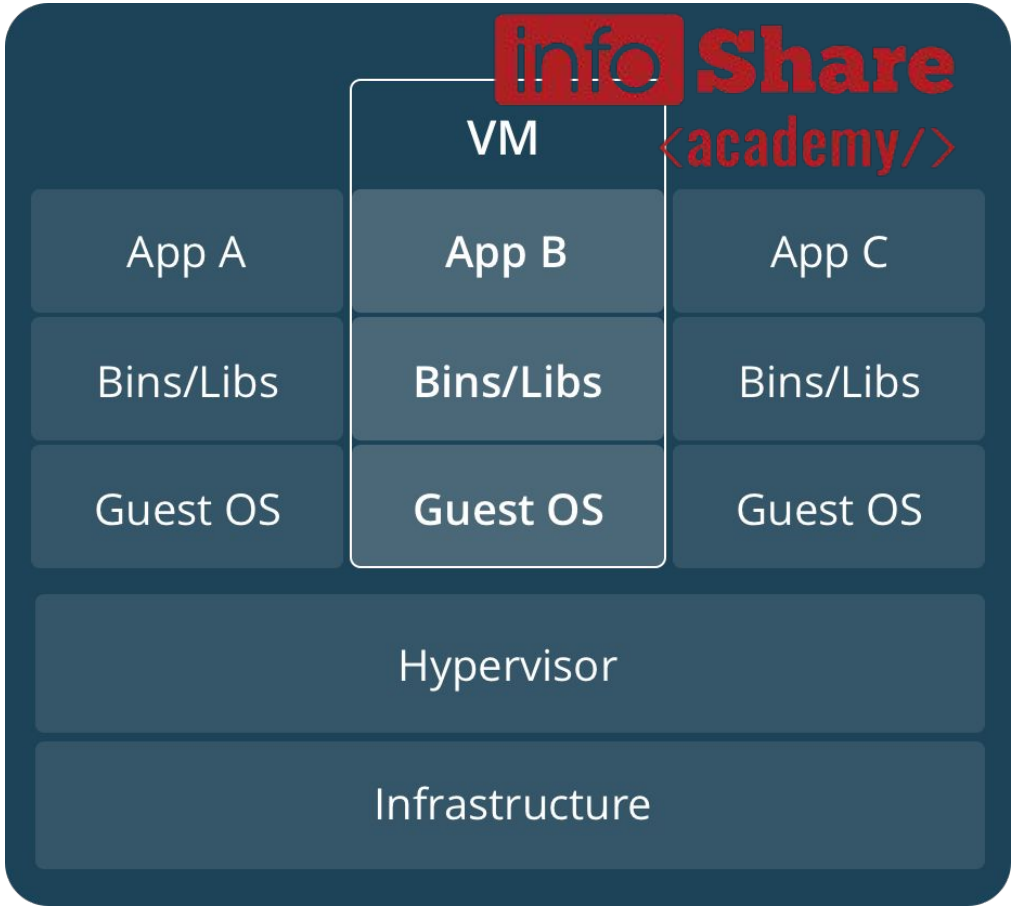


Źródło: https://pl.wikipedia.org/wiki/J%C4%85dro_systemu_operacyjnego#/media/File:Kernel-pl.svg

Zwykły serwer fizyczny

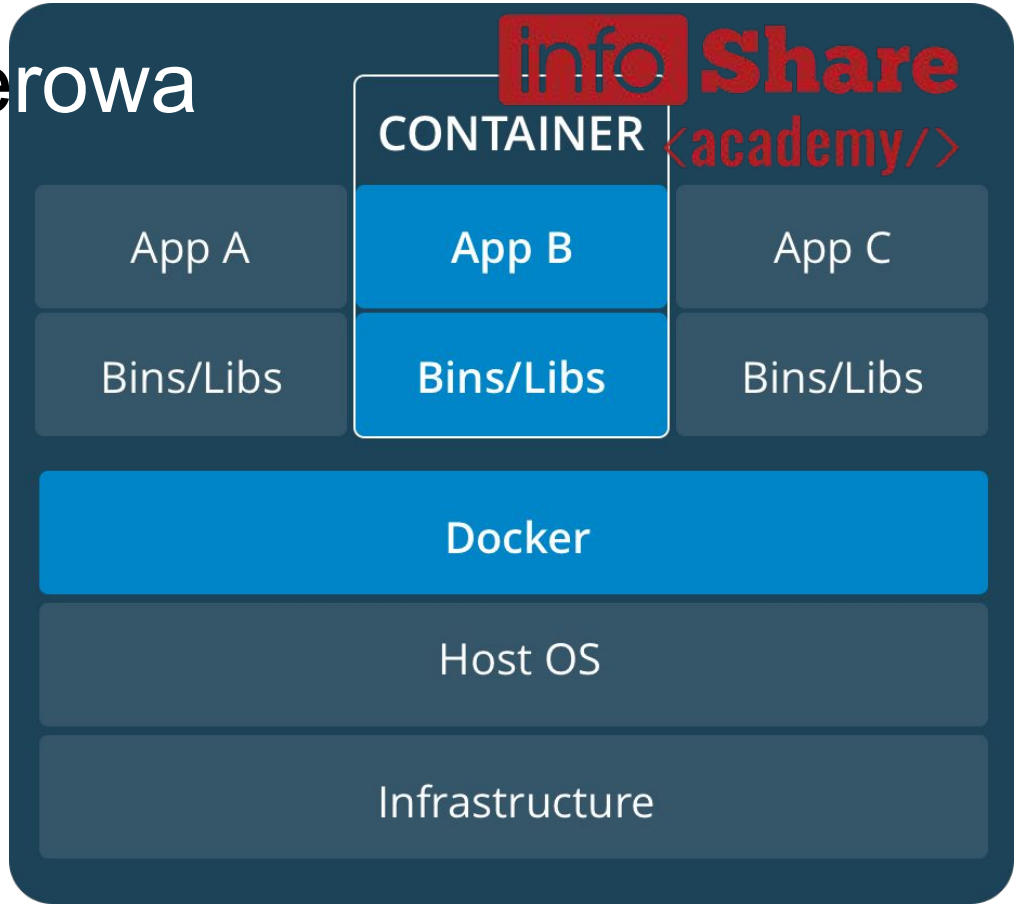
Aplikacja	Aplikacja w języku JAVA
Programy/Narzędzia	Wildfly, JRE, Postgresql, Vim, Git
System operacyjny	Ubuntu/Debian
Jądro systemu	Linux

Zwykła wirtualizacja



źródło: https://www.docker.com/

Wirtualizacja kontenerowa



źródło: https://www.docker.com/

Jak się mają zasoby?

```
#C docker stats
#c docker system df
```

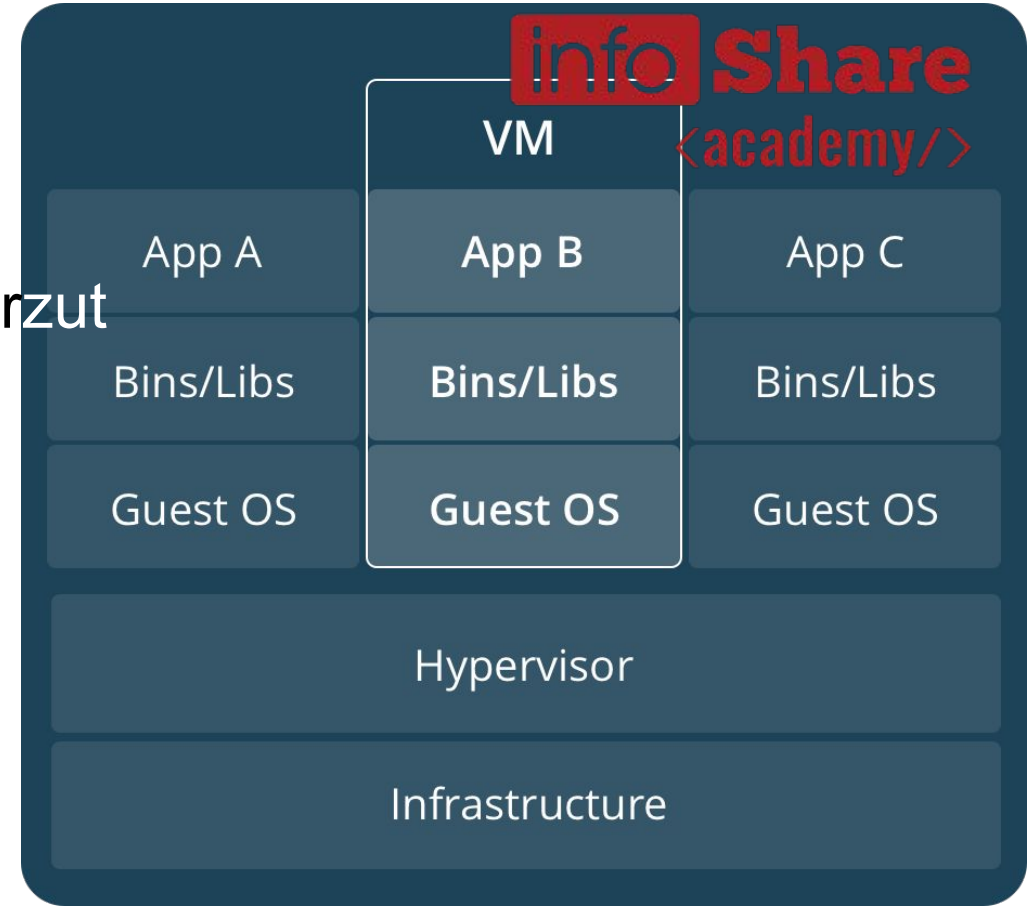
- **stats** - pokazuje jak zasoby sprzętowe typu RAM, Procesor, Sieć, Dysk są wykorzystywane
- **system** - pokazuje ile Dysku zużywamy i ile możemy odzyskać

Zwykła wirtualizacja

- uruchamia cały **system**
- **Hypervisor** generuje narzut na zasoby

\+ potrafi uruchomić różne systemy takie jak:

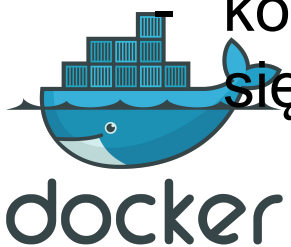
- Linux
- Windows
- OS X



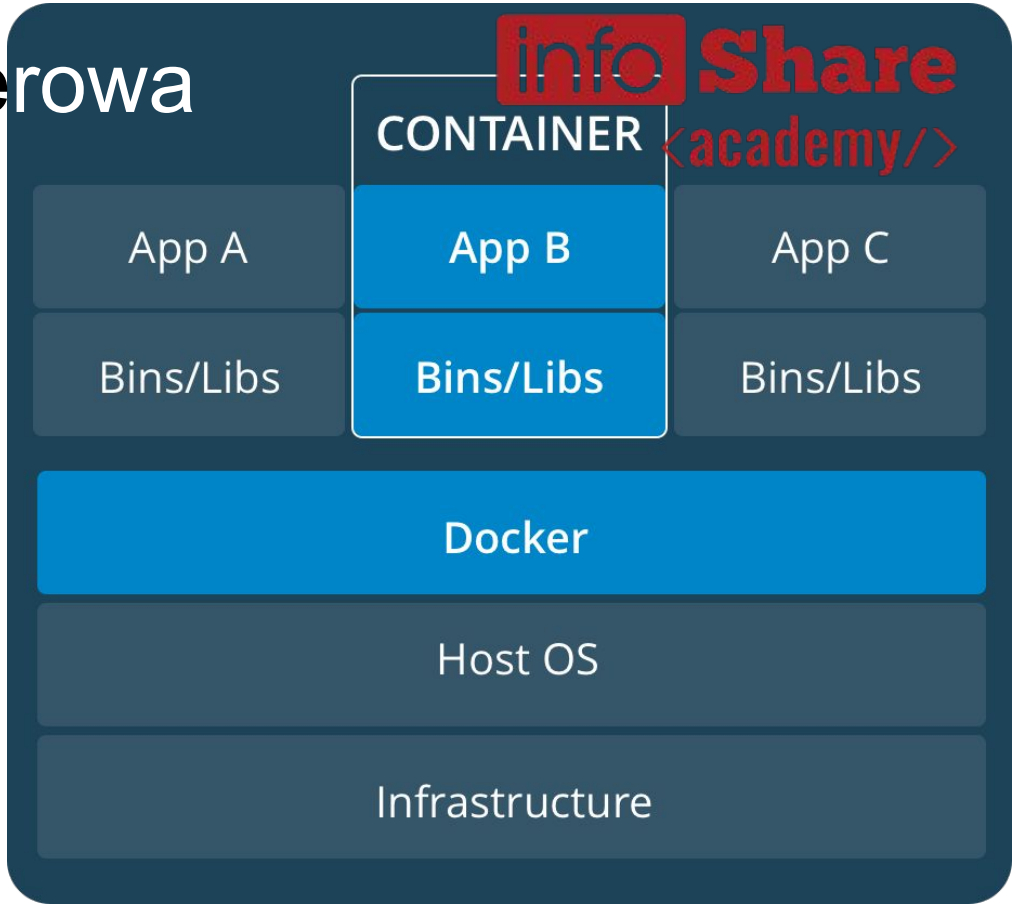
źródło: https://www.docker.com/

Wirtualizacja kontenerowa

- \+ nie generuje narzutu na uruchomienie **systemu**
- \+ korzysta z już uruchomionego, jednego **jądra systemu**
- \- kontenery uruchamia się na Linuxie



LXC



źródło: https://www.docker.com/

Zanim powstał Docker

- **2002 r.** - została wydana pierwsza wersja projektu **namespaces**, który potrafił separować/grupować:
  - nazwę serwera (hostname),
  - procesy,
  - system plików,
  - dostępu do sieci,
  - użytkowników.

  Został stworzony na poziomie **jądra systemu.**
  Napisany w języku **C.**

Zanim powstał Docker

- **2007 r.** - została wydana pierwsza wersja projektu **cgroups**, dzięki czemu na poziomie **jądra systemu**, mamy możliwość limitowania dostępu do zasobów takich jak:
  - procesor,
  - pamięć RAM,
  - przestrzeń dyskowa itp.

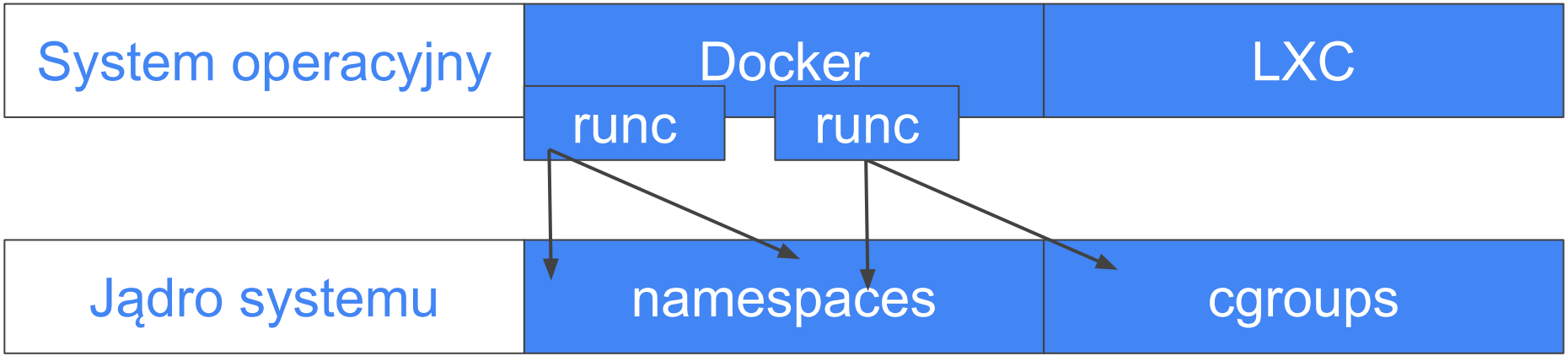
  Napisany został w języku **C.**

Zanim powstał Docker

- **2008 r.** - pierwsze wydanie projektu **LXC (Linux Containers)** wirtualizacja na poziomie **systemu operacyjnego** napisany w **C/Python/Shell**

Z czego korzysta docker?

- **2013 r. - Docker** platforma na poziomie **systemu operacyjnego** napisanego w **GO**



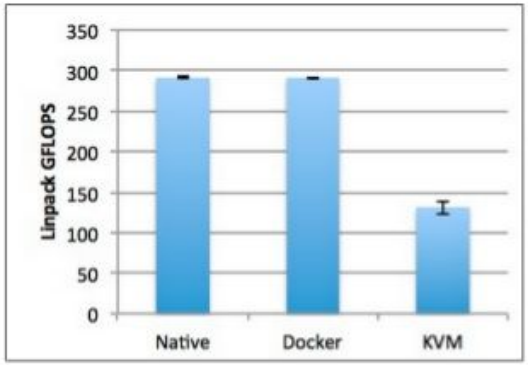
Zalety Dockera

- Wydajniejszy
- Szybszy

Docker: Performance

IBM

- Performance comparison with KVM by IBM Research shows near bare metal performance that "equals or exceeds KVM performance in every case we tested."
- See full report at: http://domino.research.ibm.com/library/cyberdig.nsf/papers/0929052195DD819C85257D2300681E7B/%24File/rc25482.pdf



IBM's results for Docker vs. KVM results running Linpack on two sockets with 16 cores.
Each data point is the arithmetic mean obtained from ten runs.
Error bars indicate the standard deviation obtained overall runs.

źródło: http://www.slideshare.net/RyanHodgin/docker-overview-rise-of-the-containers


Problemy pierwszego świata
w pracy Developera

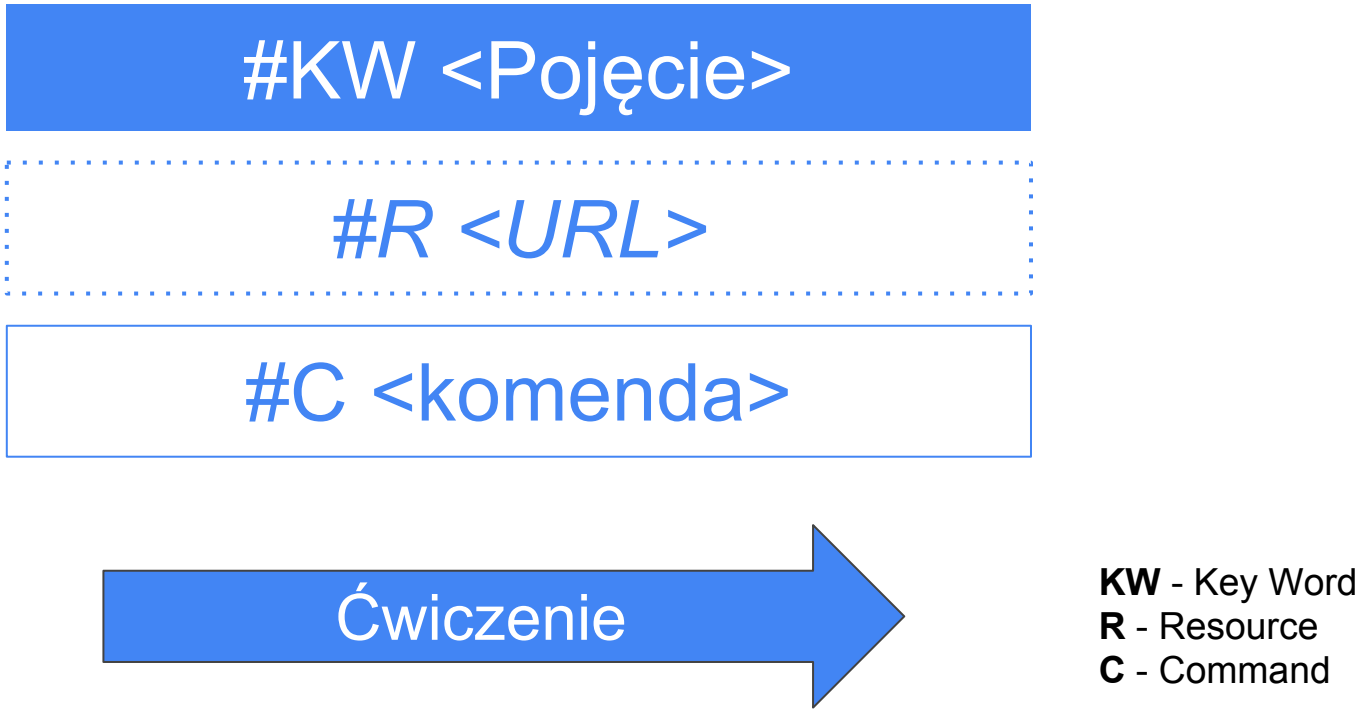
Problemy, które rozwiązuje Docker

- u mnie nie działa, a u niego tak,
- mam tylko jedną maszynę,
- mam coraz większy “śmietnik” na komputerze,
- nie mogę przetestować nowej technologii,
- nie mogę sprawdzić zainstalowanej już technologii w nowej wersji,
- tworzenie nowych “ficzerów” jest kosztowne i generuje nowy dług technologiczny,
- nowy projekt i znowu te same problemy,
- monolit czy mikroserwisy?

```
gin: Wed Mar  8 21:40:12 on ttys003
-MacBook-Air:scripts marcindargacz$ docker -v
version 1.13.1, build 092cba3
-MacBook-Air:scripts marcindargacz$ docker images
ORY          TAG                 IMAGE ID            CREATED             SIZE
             dind                7d6978320b24        7 weeks ago         99 MB
-MacBook-Air:scripts marcindargacz$ docker images
ORY          TAG                 IMAGE ID            CREATED             SIZE
             dind                7d6978320b24        7 weeks ago         99 MB
-MacBook-Air:scripts marcindargacz$ dokcer ps -a
dokcer: command not found
-MacBook-Air:scripts marcindargacz$ docker ps -a
ER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS
-MacBook-Air:scripts marcindargacz$ dokcer pull ubuntu
dokcer: command not found
-MacBook-Air:scripts marcindargacz$ docker pull ubuntu
efault tag: latest
 Pulling from library/ubuntu
db41d: Extracting [============                                      ==========>] 50.43 MB/50.43 MB
45a87: Download complete
7c39f: Download complete
e9079: Download complete
9eaaa: Download complete
```

Zaczynamy !!!

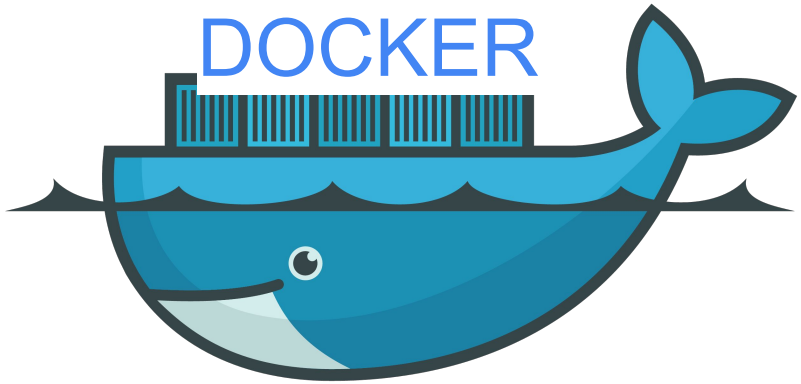
Podstawowe oznaczenia



KW - Key Word
R - Resource
C - Command

infoShare academy

PODSTAWY NARZĘDZIA DOCKER



docker

Dokumentacja

```
#C docker -h
```

Dokumentacja

#R https://docs.docker.com

Obraz

Obraz

#KW Image

- może zawierać zainstalowane odpowiednie biblioteki, aplikacje,
- posiada zdefiniowaną konfigurację środowiska
- składa się z warstw: **docker history <nazwa_obrazu>**

Lista obrazów

```
#C docker images
#C docker image ls
```

Rejestr obrazów

#KW Registry

Rejestr obrazów

#KW Registry

- przechowuje obrazy
- dystrybuuje obrazy
- nazwa repozytorium + tag = obraz

```
jboss/wildfly       latest              d4b4d01b53bd
```

Rejestr obrazów

Publiczny rejestr

#R https://hub.docker.com

Lokalny rejestr

#C docker images

Własny prywatny rejestr

#R https://docs.docker.com/registry/

Pobieranie obrazu

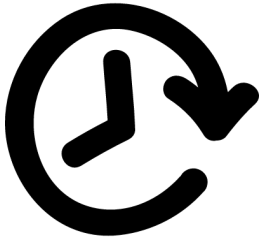
```
#C docker pull <nazwa_repo>:[tag]
```

Ćwiczenie nr 1

1. Pobrać najnowszy obraz **debian** z publicznego repozytorium
2. Wyświetlić pobrany obraz w konsoli
3. * Pobrać również **8** wersję **debiana**

Rozwiązanie nr 1

1. docker pull debian
2. docker images
3. docker pull debian:8



Retrospekcja - czego się nauczyliśmy?

infoShare academy

1. Poznaliśmy historię i korzenie **Dockera**
2. Wiemy czym się różni **Docker** od **zwykłej wirtualizacji**
3. Wiemy czym jest **rejestr** a czym **repozytorium**
4. Wiemy czym jest **obraz** i jak go pobrać z **publicznego** do **lokalnego rejestru**

Kontener

#KW Container

Kontener

#KW Container

- tworzy i uruchamia się na podstawie **obrazu,**
- z jednego **obrazu** można uruchomić wiele **kontenerów**,
- **kontener** może być zadaniowy (wykonuje swoje zadanie, a potem się “wyłącza”)
- **kontener** może działać bez przerwy

Lista kontenerów

```
#C docker ps
#C docker container ls
```

Uruchamianie kontenera

```
#C docker run <nazwa_repo|id_obrazu>[:tag]
```

Jeżeli podana nazwa **obrazu** nie istnieje w **lokalnym rejestrze** to spróbuje go pobrać z **publicznego rejestru**, a następnie zostanie uruchomiony **kontener**

Lista kontenerów

```
#C docker ps
```

PRACUJĄCYCH!

Lista wszystkich kontenerów

```
#C docker ps -a
```

Uruchomienie kontenera z przekazaniem zmiennej środowiskowej

```
#C docker run -it -e [nazwa_zmiennej_srodowiskowej]=[wartość_zmiennej] <nazwa_repo|id_obrazu> [komenda]
```

zmienna środowiskowa = własny parametr

Zarządzanie kontenerem

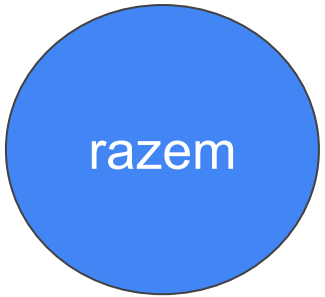
```
#C docker stop
```

```
#C docker start
```

```
#C docker kill
```

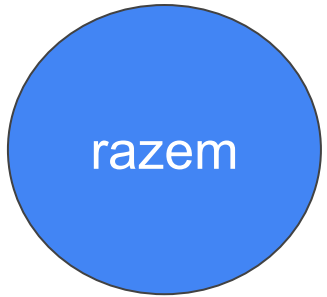
Jako parametr podajemy nazwę **kontenera** lub id **kontenera**

Ćwiczenie nr 2



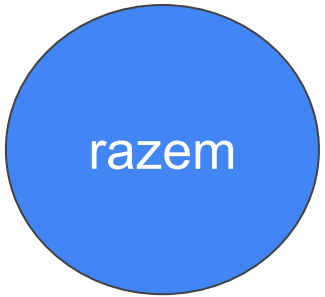
1. Uruchomić kontener z serwerem MySQL
2. Zalogować się i wykonać zapytanie SQL

Rozwiązanie



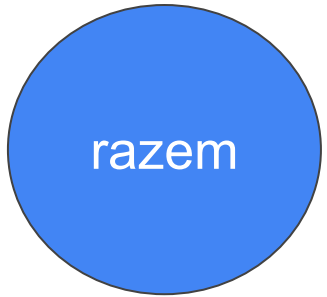
1. Odwiedzamy stronę https://hub.docker.com i szukamy **obrazu** MySQL

Rozwiązanie



1. Odwiedzamy stronę https://hub.docker.com i szukamy **obrazu** MySQL
2. docker run -d -e MYSQL_ROOT_PASSWORD=root mysql:8

Rozwiązanie



1. Odwiedzamy stronę https://hub.docker.com i szukamy **obrazu** MySQL
2. docker run -d -e MYSQL_ROOT_PASSWORD=root mysql:8
3. docker exec -it <id_kontenera> mysql -uroot -proot

Uruchomienie komendy na kontenerze

```
#C docker exec -it <nazwa/id_kontenera>
<komenda>
```

-it : daje interakcję z **kontenerem**
(przechwytuje proces)
np. docker exec -it debian bash

Logi kontenera

```
#C docker logs -f <nazwa/id_kontenera>
```

-f daje możliwość nasłuchiwania logów

Operacje globalne

```
#C docker stop $(docker ps -q)
```

Zatrzymanie wszystkich aktywnych **kontenerów**

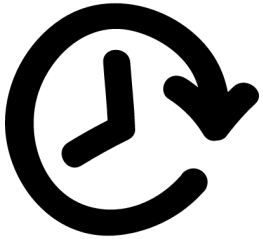
```
#C docker rm $(docker ps -a -q)
docker container prune
```

Usunięcie wszystkich **kontenerów**

Operacje globalne

```
#C docker rmi $(docker images -q)
```

Usunięcie wszystkich **obrazów**



Retrospekcja - czego się nauczyliśmy?

1. Wiemy czym jest **kontener**
2. Potrafimy znaleźć i uruchomić interesujący nas **kontener** z **obrazu**
3. Potrafimy zatrzymać/wznowić/”zabić” **kontener**
4. Potrafimy podejrzeć logi działającego **kontenera**
5. Znamy kilka ciekawych globalnych operacji na **kontenerach** i **obrazach**

infoShare academy

WŁASNE OBRAZY

docker

Definicja obrazu

#KW Dockerfile

Definicja obrazu

#KW Dockerfile

```
FROM jboss/wildfly

MAINTAINER "InfoShare Academy"

RUN mkdir -p wildfly/modules/system/layers/base/com/mysql/driver/main
ADD config/mysql.module.xml wildfly/modules/system/layers/base/com/mysql/driver/main/module.xml
ADD config/mysql-connector-java-5.1.42-bin.jar wildfly/modules/system/layers/base/com/mysql/driver/main/

RUN wildfly/bin/add-user.sh admin admin --silent

ADD target/api.war wildfly/standalone/deployments/

CMD ["/opt/jboss/wildfly/bin/standalone.sh", "-b", "0.0.0.0", "-bmanagement", "0.0.0.0"]
```

← Przykładowy plik **Dockerfile**

Definicja obrazu

#KW Dockerfile

```
FROM jboss/wildfly

MAINTAINER "InfoShare Academy"

RUN mkdir -p wildfly/modules/system/layers/base/com/mysql/driver/main
ADD config/mysql.module.xml wildfly/modules/system/layers/base/com/mysql/driver/main/module.xml
ADD config/mysql-connector-java-5.1.42-bin.jar wildfly/modules/system/layers/base/com/mysql/driver/main/

RUN wildfly/bin/add-user.sh admin admin --silent

ADD target/api.war wildfly/standalone/deployments/

CMD ["/opt/jboss/wildfly/bin/standalone.sh", "-b", "0.0.0.0", "-bmanagement", "0.0.0.0"]
```

← Przykładowy plik **Dockerfile**

Przy użyciu specjalnych słów kluczowych możemy zdefiniować własny obraz

Definicja obrazu
podstawowe słowa kluczowe

infoShare academy

#KW **FROM**

#KW **MAINTAINER**

#KW **RUN**

#KW **COPY**

#KW **CMD**

Definicja obrazu
podstawowe słowa kluczowe

infoShare academy

#KW FROM

Nazwa **repozytorium,** na którym bazujemy
np. **FROM** jboss/wildfly

Definicja obrazu
podstawowe słowa kluczowe

#KW MAINTAINER

Nazwa właściciela obrazu
np. **MAINTAINER** “InfoShare Academy”

Definicja obrazu
podstawowe słowa kluczowe

Uruchomienie komendy na **obrazie**
np. **RUN** mkdir /app

#KW **RUN**

Definicja obrazu
podstawowe słowa kluczowe

Kopiowanie z **hosta** na **kontener**
np. **COPY** target/app.war /jboss/app/app.war

#KW COPY

Definicja obrazu
podstawowe słowa kluczowe

Komenda, która ma być uruchomiona przy starcie **kontenera**
np. **CMD** [“/bin/standalone.sh”, “-b”, “0.0.0.0”]

#KW **CMD**

Lokalne budowanie obrazu

```
#C docker build .
```

Komenda musi być wykonana w miejscu, gdzie znajduje się plik **Dockerfile**

Lokalne budowanie obrazu

```
#C docker build -t isa/docker:0.1 .
```

Nadajemy nazwę **repozytorium** oraz **tag**
W tym przypadku:
repozytorium = isa/docker
tag = 0.1

Ćwiczenie nr 3

branch: dockerfile

1. Zbudować własny obraz z pliku Dockerfile

Lokalne budowanie obrazu

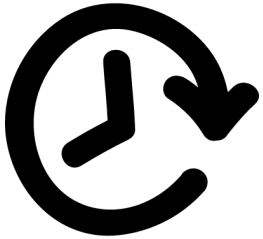
```
#C docker build -t isa/docker:0.1 -t
isa/docker:latest .
```

Tworzenie wielu **tagów**

Ćwiczenie nr 4

branch: dockerfile_java

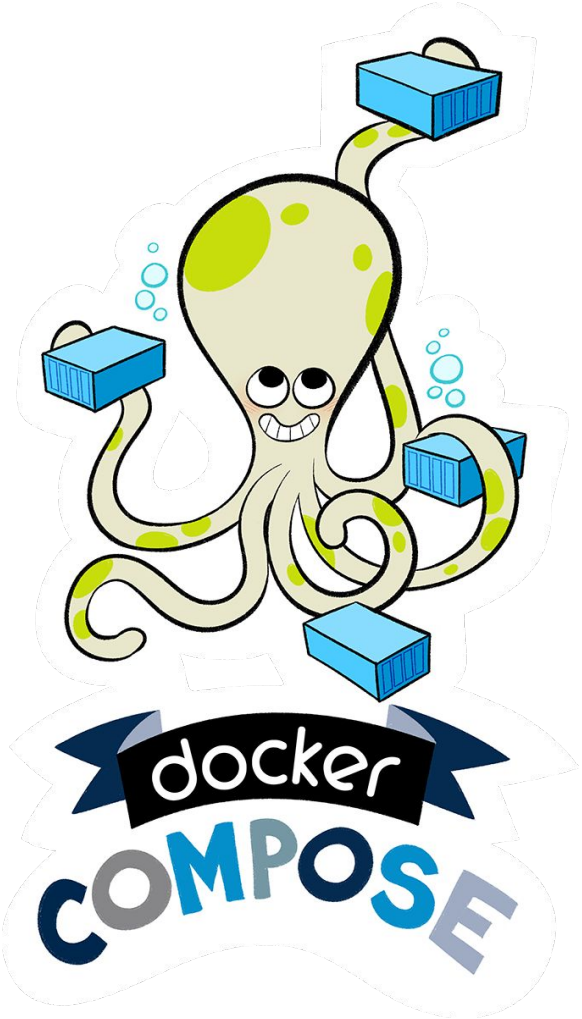
1. Stworzyć obraz, który osadzi paczkę **war** na **wildfly**



Retrospekcja - czego się nauczyliśmy?

infoShare academy

1. Do czego służy plik **Dockerfile**
2. Jak zbudować własny **obraz**



infoShare
<academy/>

Docker compose

#KW docker-compose

Docker compose

infoShare academy

#KW docker-compose

- Pozwala uruchamiać i zarządzać wieloma **kontenerami**
- Można definiować zależności między **kontenerami**
- Można mapować porty **hosta** i konkretnego **kontenera**
- Można zamykać **kontenery** w grupy sieciowe

Docker compose

infoShare academy

#KW docker-compose

```yaml
version: '3'
services:
  api:
    build: ../api/
    ports:
      - 8080:8080
      - 9990:9990
    volumes:
      - ../api/target/api.war:/opt/jboss/wildfly/standalone/deployments/api.war
    networks:
      - frontend
      - backend
    depends_on:
      - db
  db:
    image: mysql:8
    ports:
      - 3306:3306
    networks:
      - backend
    environment:
      - MYSQL_ROOT_PASSWORD=root

networks:
  frontend:
    driver: bridge
  backend:
    driver: bridge
```

Wiele **kontenerów** definiuje się w pliku **docker-compose.yml**

← przykładowy plik **docker-compose.yml**

Serwisy

#KW service

Serwisy

#KW service

x razy ten sam kontener = serwis

Budowanie wszystkich obrazów

```
#C docker-compose build
```

Uruchamianie wszystkich serwisów

```
#C docker-compose up -d
```

```
#C docker-compose up --build -d
```

Zarządzanie serwisami

infoShare academy

```
#C docker-compose stop
```

```
#C docker-compose start
```

```
#C docker-compose kill
```

```
#C docker-compose logs
```

Ćwiczenie nr 5

branch: docker_compose

branch: docker_compose_extra

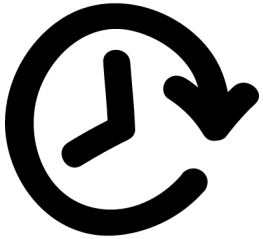
1. Uruchomić usługi zdefiniowane w **docker_compose.yml**

branch: docker_compose - zawiera uruchomienie app + db
branch: docker_compose_extra - zawiera uruchomienie app + panel Wildfly + db

Ćwiczenie nr 6

branch: docker_compose_mysql

1. Uruchomić gotową infrastrukturę, która przyda się do projektu
2. Pokazać, że jest połączenie aplikacji z DB (**UWAGA!** Jeżeli nie działa należy usunąć usługi i zbudować je na nowo)



Retrospekcja - czego się nauczyliśmy?

1. Czym jest **serwis**
2. Podstawowe możliwości narzędzia **docker-compose**
3. Czym jest **docker-compose.yml**
4. Jak budować i zarządzać systemem zbudowanym z wielu **serwisów**

Przegląd dodatkowych narzędzi

Graficzne interfejsy dla obrazów
i kontenerów

Kitematic - jako aplikacja Desktop

Portainer - jako aplikacja WEB

Przegląd dodatkowych narzędzi

Docker w chmurze

- **Docker cloud**:
  - własne registry (1 prywatne + nieskończoność publicznych) (chyba, że zapłacisz)
  - automatyczne budowanie

linkedin.com/in/mdargacz

marcin.dargacz@gmail.com

Dziękuję za uwagę!