info Share ACADEMY

Docker/VM/emulatory

Witajcie !

Przemysław Grzesiowski

Przygotowanie do zajęć

1. http://isoredirect.centos.org/centos/7/isos/x86_64/CentOS-7-x86_64-DVD-1804.iso
2. docker działa bez sudo (docker -v)
3. docker pull jenkins/jenkins
4. docker pull mongo

Agenda

1. Wstęp
2. Rozwinięcie
3. Zakończenie

Agenda

1. Wstęp
2. Emulatory
3. Wirtualne maszyny
4. Kontenery
5. Docker
    a. Kontenery
    b. Obrazy
    c. Wolumeny
    d. Sieci
    e. CI/CD
    f. Budowa własnych obrazów
    g. Wady dockera

Virtual machine ?

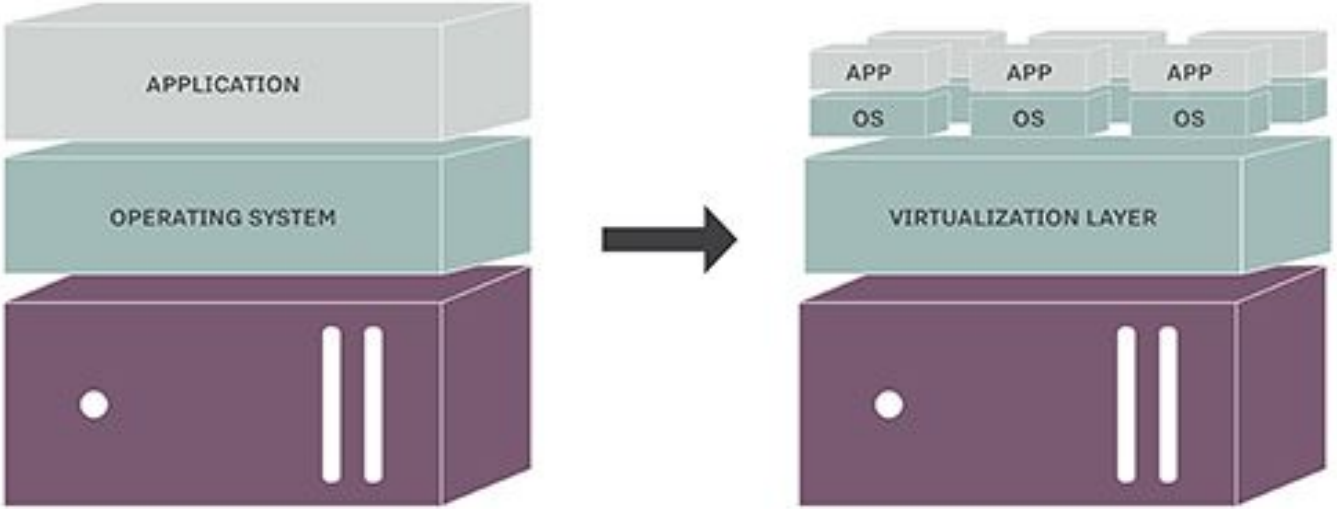
Virtual machine

“ *an efficient, isolated duplicate of a real computer machine.”*

Gerald J. Popek, Robert P. Goldberg 1974

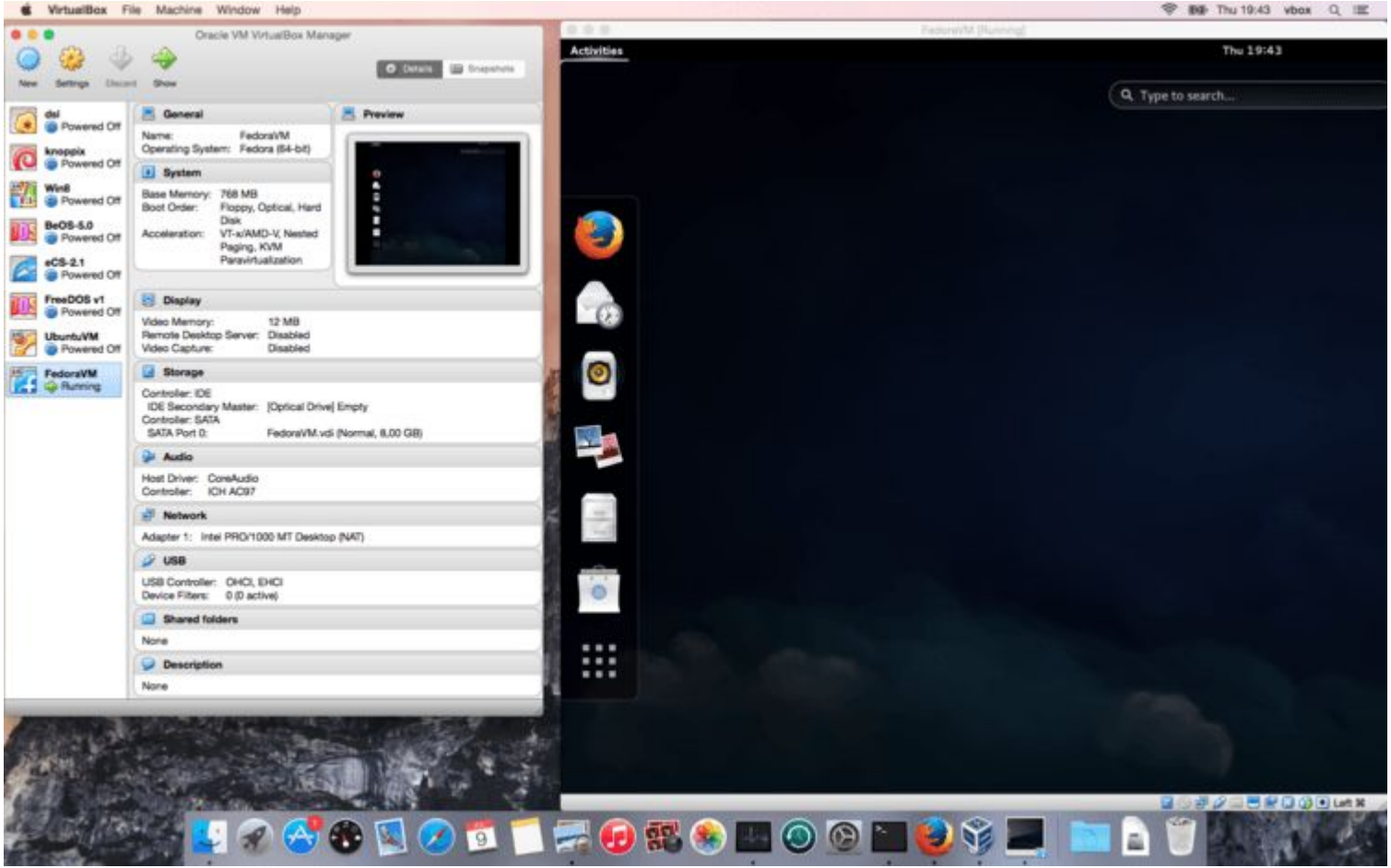
infoShare ACADEMY

TRADITIONAL AND VIRTUAL ARCHITECTURE

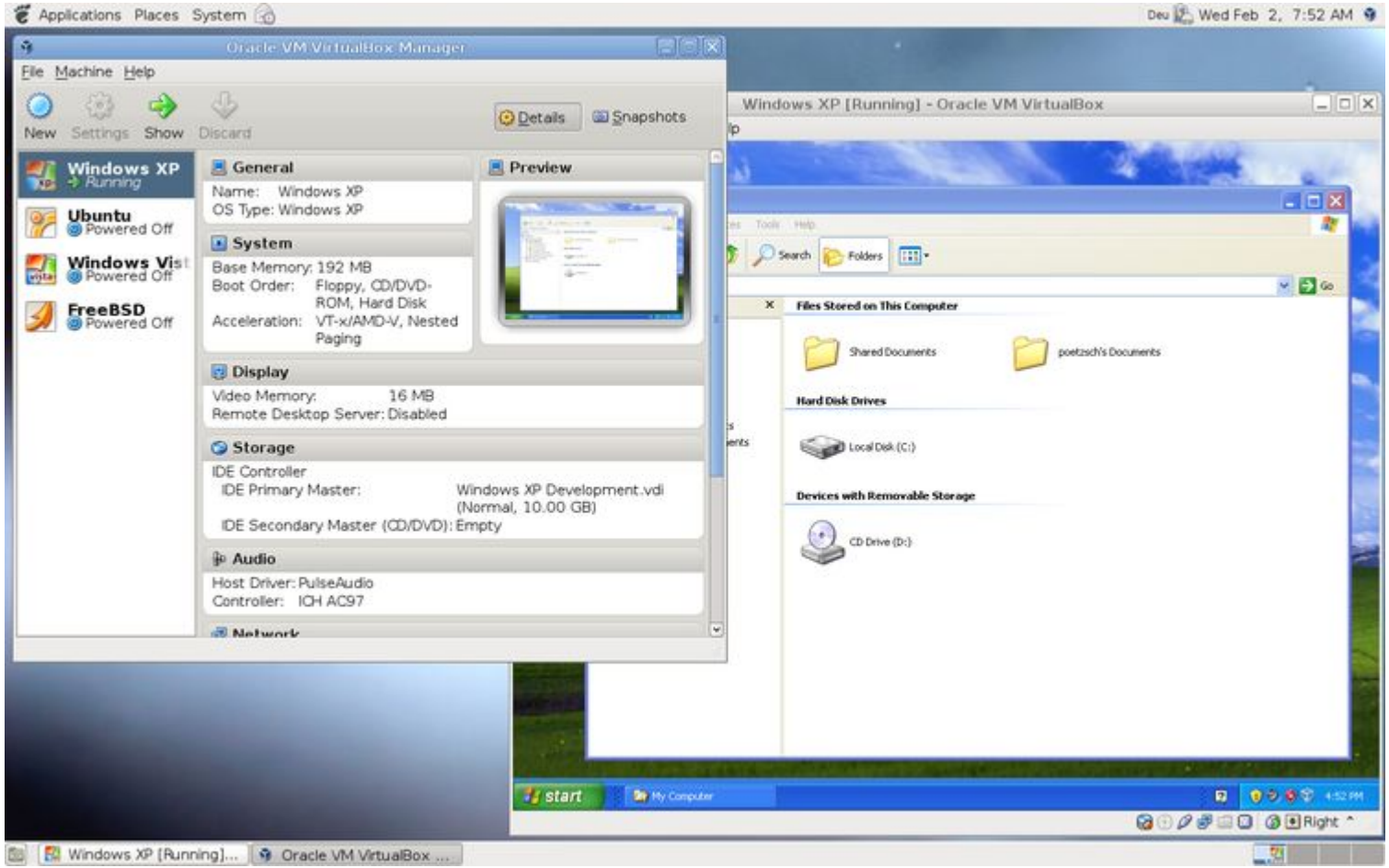


TRADITIONAL ARCHITECTURE

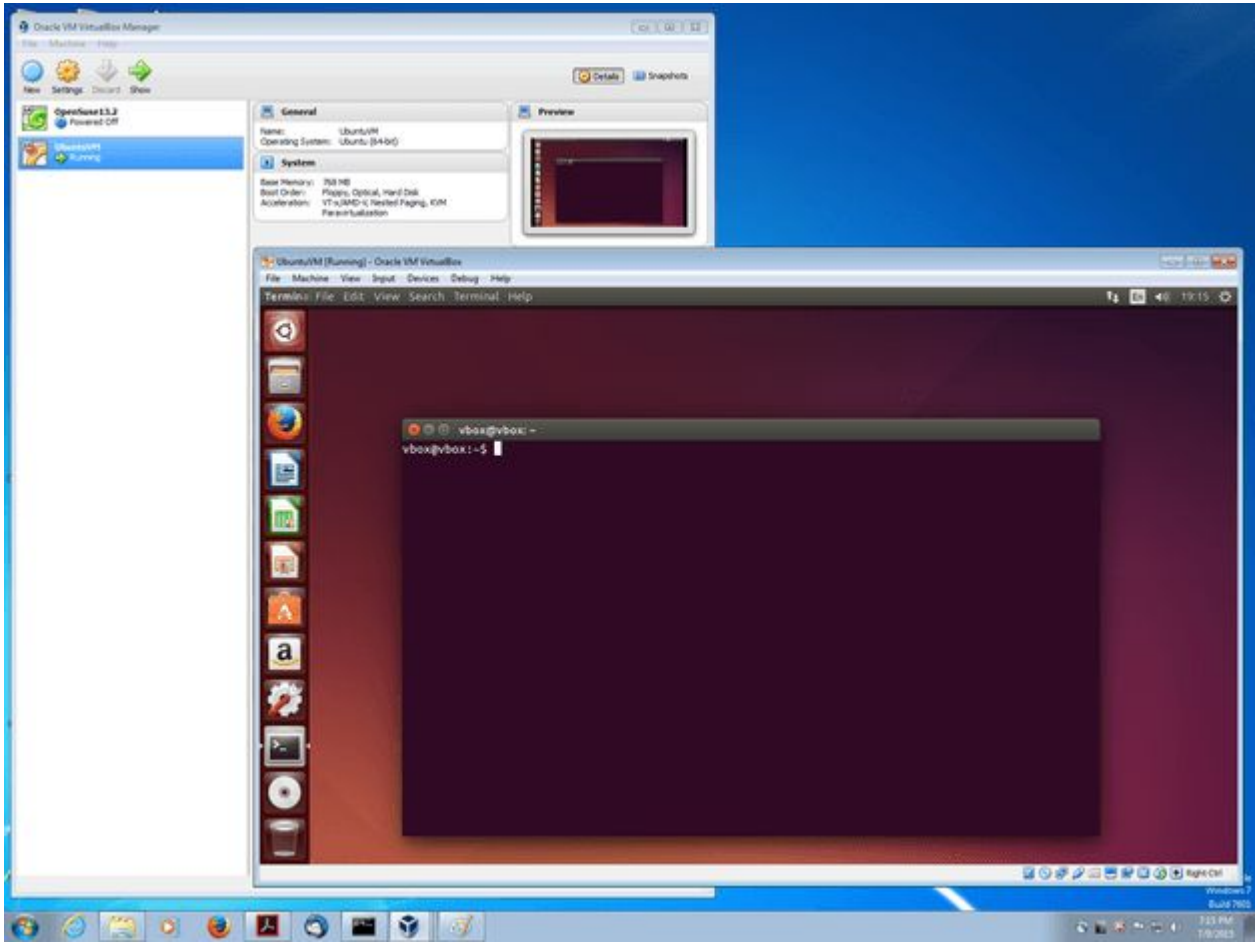
VIRTUAL ARCHITECTURE



Fedora 21 on OSX



Linux running a Windows XP


Oracle VM VirtualBox Manager
New
Settings
Discard
Show
Details
Snapshots
OpenSuse13.2
Powered Off
UbuntuVM
Running
General
Name: UbuntuVM
Operating System: Ubuntu (64-bit)
System
Base Memory: 768 MB
Boot Order: Floppy, Optical, Hard Disk
Acceleration: VT-x/AMD-V, Nested Paging, KVM Paravirtualization
Preview
UbuntuVM [Running] - Oracle VM VirtualBox
File Machine View Input Devices Debug Help
Terminal File Edit View Search Terminal Help
vbox@vbox: ~
vbox@vbox:~$

Różne smaki maszyny wirtualnej

System VM
(full virtualization)

* provide a substitute for a real machine
* VM simulates enough hardware to allow an unmodified "guest" OS to be run in isolation
* the guest operating system "thinks" it's running on real machine

* *Hyper-V, virtual Box, VMware, virtual PC*

Różne smaki maszyny wirtualnej

System VM (full virtualization)

* provide a substitute for a real machine
* VM simulates enough hardware to allow an unmodified "guest" OS to be run in isolation
* the guest operating system "thinks" it's running on real machine

* *Hyper-V, virtual Box, VMware, virtual PC*

Process VM

* designed to execute computer programs in a platform- independent environment

* supports single process/app

* *Java Virtual Machine (JVM), .NET Framework*

Różne smaki maszyny wirtualnej

System VM (full virtualization)

* provide a substitute for a real machine
* VM simulates enough hardware to allow an unmodified "guest" OS to be run in isolation
* the guest operating system "thinks" it's running on real machine

* *Hyper-V, virtual Box, VMware, virtual PC*

Process VM

* designed to execute computer programs in a platform- independent environment

* supports single process/app

* *Java Virtual Machine (JVM), .NET Framework*

operating-system -level virtualization (Containerization)

* the same kernel is used by guest system (almost zero performance overhead comparing to full VM)

* *docker, FreeBSD jail, Solaris Containers, OpenVZ*

1. Emulatory

EMULACJA

“Naśladowanie” całego środowiska – hardware + software.

PO CO TO?


DO KINA? PO CO?
MATI, ŚCIONGNIJ Z TEGO DORENTA
MATKA NAROBI POPKORNU
BLASTY.PL

PO CO TO?

Emulator iPhone – 0 PLN
iPhone X - ~5200 PLN

Emulator – ok. 10 kliknięć
iPhone X – znaleźć najlepszą ofertę, zamówić, zapłacić, czekać …

Gdzie jest haczyk?

PLUSY	MINUSY
Tanie - darmowe	
Proste – ściągamy, instalujemy i działamy!	
Integracja – emulatory łatwiej zsynchronizować z IDE niż fizyczne urządzenie	

Gdzie jest haczyk?

PLUSY	MINUSY
Tanie - darmowe	**Symulowanie** – emulator nie potrafi symulować np. zużycia baterii, czy przerwań spowodowanych czynnikami zew.
Proste – ściągamy, instalujemy i działamy!	**Wyświetlanie** – jakość będzie inna niż na rzeczywistym urządzeniu
Integracja – emulatory łatwiej zsynchronizować z IDE niż fizyczne urządzenie	**Pamięć** – możemy przyznać emulatorowi więcej pamięci, co fałszuje zachowanie rzeczywistego urządzenia

Przykłady emulatorów androida

Genymotion:

https://www.youtube.com/watch?v=VpKEtnO7yHc

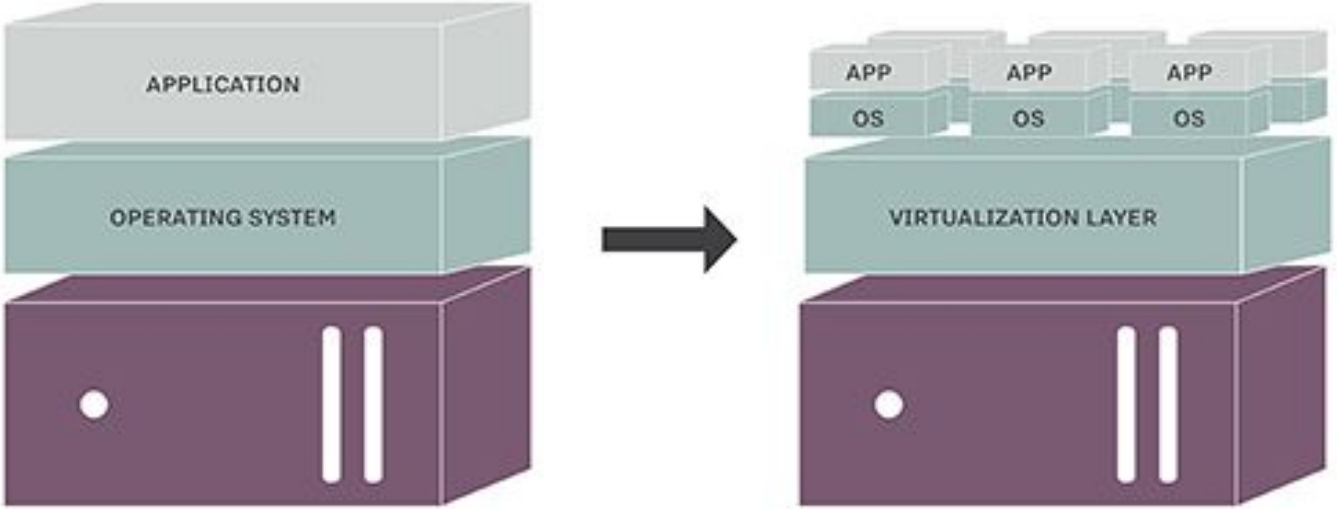
Android Studio

https://www.youtube.com/watch?v=547DXRq8zAo

2. Maszyna wirtualna

wirtualizacja

TRADITIONAL AND VIRTUAL ARCHITECTURE



TRADITIONAL ARCHITECTURE

VIRTUAL ARCHITECTURE

Hypervisor


App
App
App
Binaries/ Libraries
Binaries/ Libraries
Binaries/ Libraries
Guest OS
Guest OS
Guest OS
Hypervisor
Host OS
Server Hardware

Przykłady


ORACLE
VirtualBox


Microsoft
Hyper-V


vmware


QEMU


Parallels

Ćw.2.1.- Virtual Box

- stwórz maszynę wirtualną w oparciu o obraz CentOs7 DVD ISO

  (https://www.centos.org/download/, (“SOFTWARE SELECTION” - Gnome desktop)
- odpal maszynę wirtualną, poćwicz przełączanie się pomiędzy VM a hostem
- zajrzyj do ustawień - zwiększ ilość RAMu i rdzeni CPU, zrestartuj
- zwiększ rozdzielczość
- sprawdź czy Centos na dostęp do internetu
- uzyskaj dostęp do pendrive
- poćwicz różne wyłączania VM (power off, save state etc.)
- (*)kopiowanie zawartości schowka - w obie strony (https://linuxconfig.org/how-to-install-virtualbox-guest-additions-on-centos-7-linux)
- (*)udostępnij folder hosta w trybie do zapisu

“

virtual machine (VM) is an emulation of a computer system

info Share ACADEMY

VM - za może przeciw?

PLUS	MINUS
Koszty – zminimalizowane koszty hardware'u, energii elektrycznej, okablowania itp.	
Wykorzystanie zasobów – wykorzystujemy zasoby hardware'owe do maksimum	
Sandbox – psujemy ile chcemy! Robimy co chcemy nie patrząc na konsekwencje!	
Wielordzeniowe procesory – wirtualizacja efektywnie wykorzystuje rdzenie procesora	

Więcej: https://www.computerworld.pl/news/Za-i-przeciw-wirtualizacji,403362.html

VM - za może przeciw?

PLUS	MINUS
Koszty – zminimalizowane koszty hardware'u, energii elektrycznej, okablowania itp.	**Koszty** – licencje i hardware (przy przenoszeniu aktualnych rozwiązań)
Wykorzystanie zasobów – wykorzystujemy zasoby hardware'owe do maksimum	**Wsparcie** – nie możemy wirtualizować wszystkiego; producenci niektórych rozwiązań nie wspierają wirtualizacji
Sandbox – psujemy ile chcemy! Robimy co chcemy nie patrząc na konsekwencje!	**Opóźnienie** – wirtualne maszyny działają wolniej
Wielordzeniowe procesory – wirtualizacja efektywnie wykorzystuje rdzenie procesora	**Zarządzanie i bezpieczeństwo** – trudne i skomplikowane metody

Więcej: https://www.computerworld.pl/news/Za-i-przeciw-wirtualizacji,403362.html

3. Kontenery

Co to właściwie jest ten kontener?

Kontener “kuchnia”

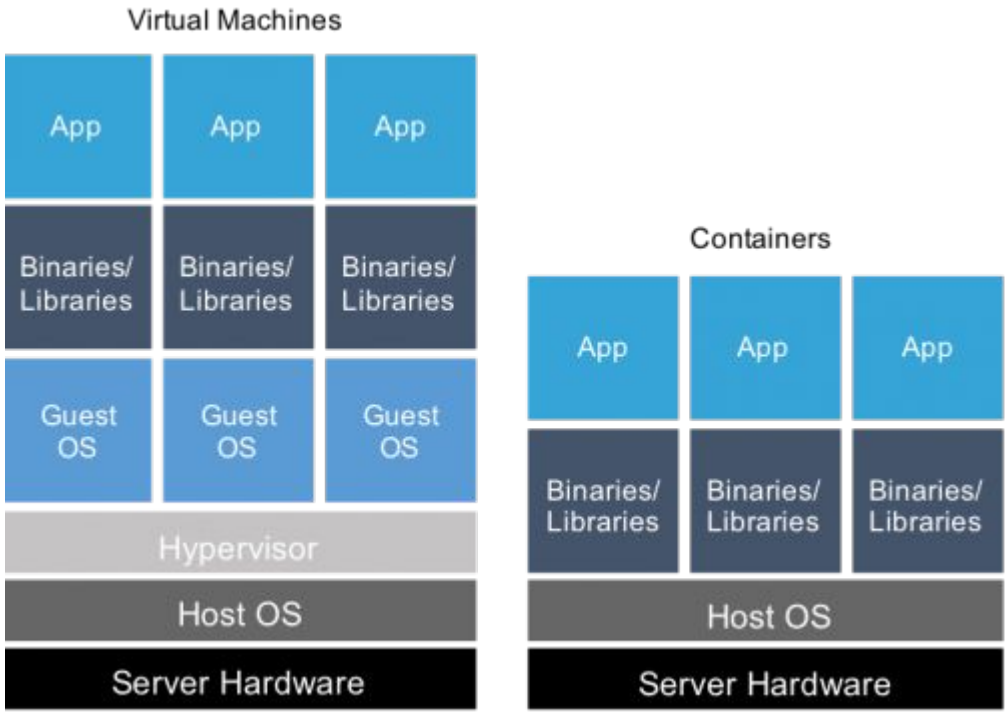
Kontenery

- OS containers can be easily imagined as a Virtual Machine (VM), but unlike a VM they share the kernel of the host operating system;

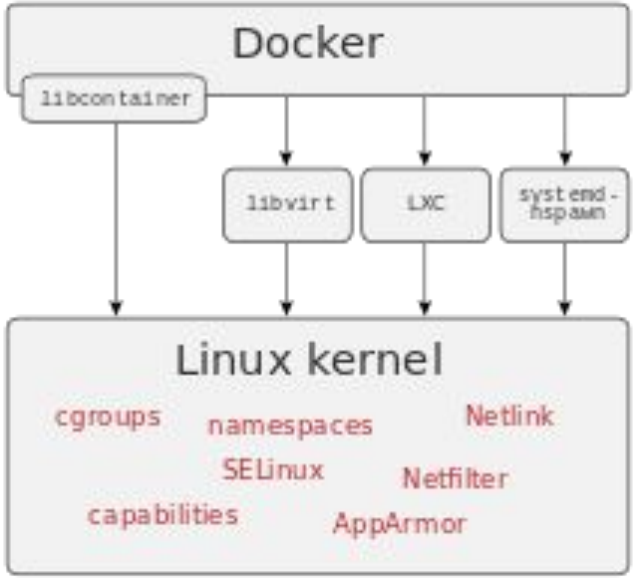
Kontenery

- OS containers can be easily imagined as a Virtual Machine (VM), but unlike a VM they share the kernel of the host operating system;
- we can install, configure, and run different applications, libraries etc. (just as you would run on any VM);

Kontener vs maszyna wirtualna



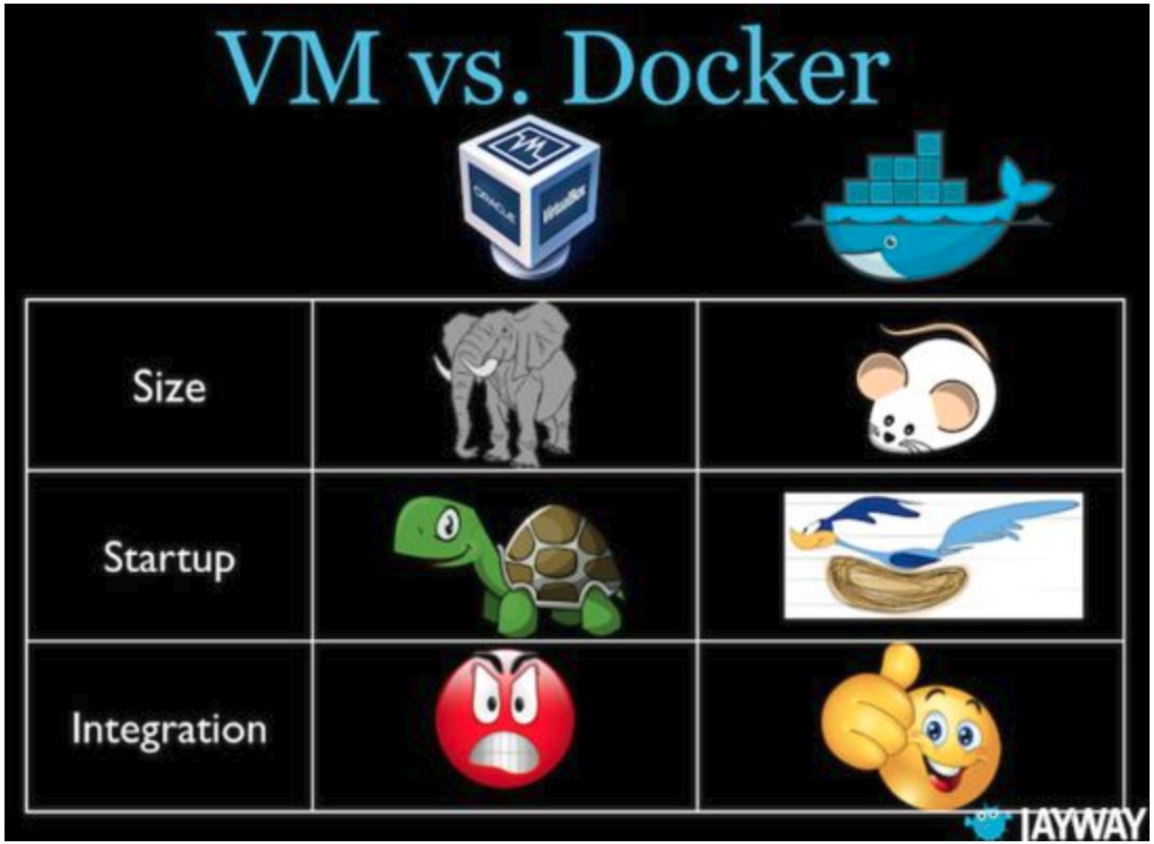
Technologie które umożliwiły powstanie dockera



- namespaces
- control groups (cgroups)
- union file-system
- container format (libcontainer)

Porównanie

Virtual Machines (VMs)	Containers
Represents hardware-level virtualization	Represents operating system virtualization
Heavyweight	Lightweight
Slow provisioning	Real-time provisioning and scalability
Limited performance	Native performance
Fully isolated and hence more secure	Process-level isolation and hence less secure



http://blog.vacha-design.com/docker-introduction/

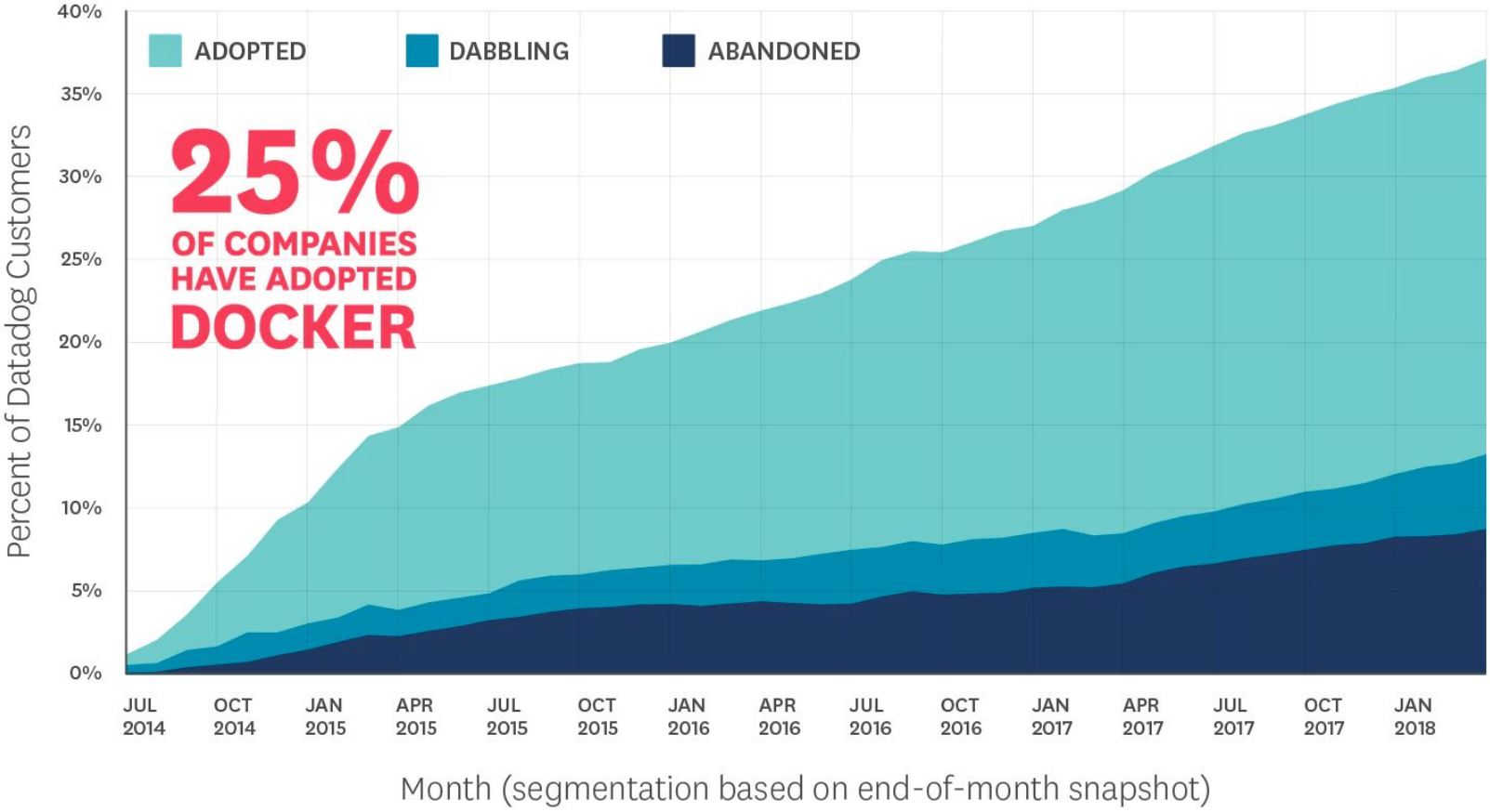

SAY ONE MORE TIME
WORKS ON MY MACHINE
memegenerator.net

Dlaczego Docker?

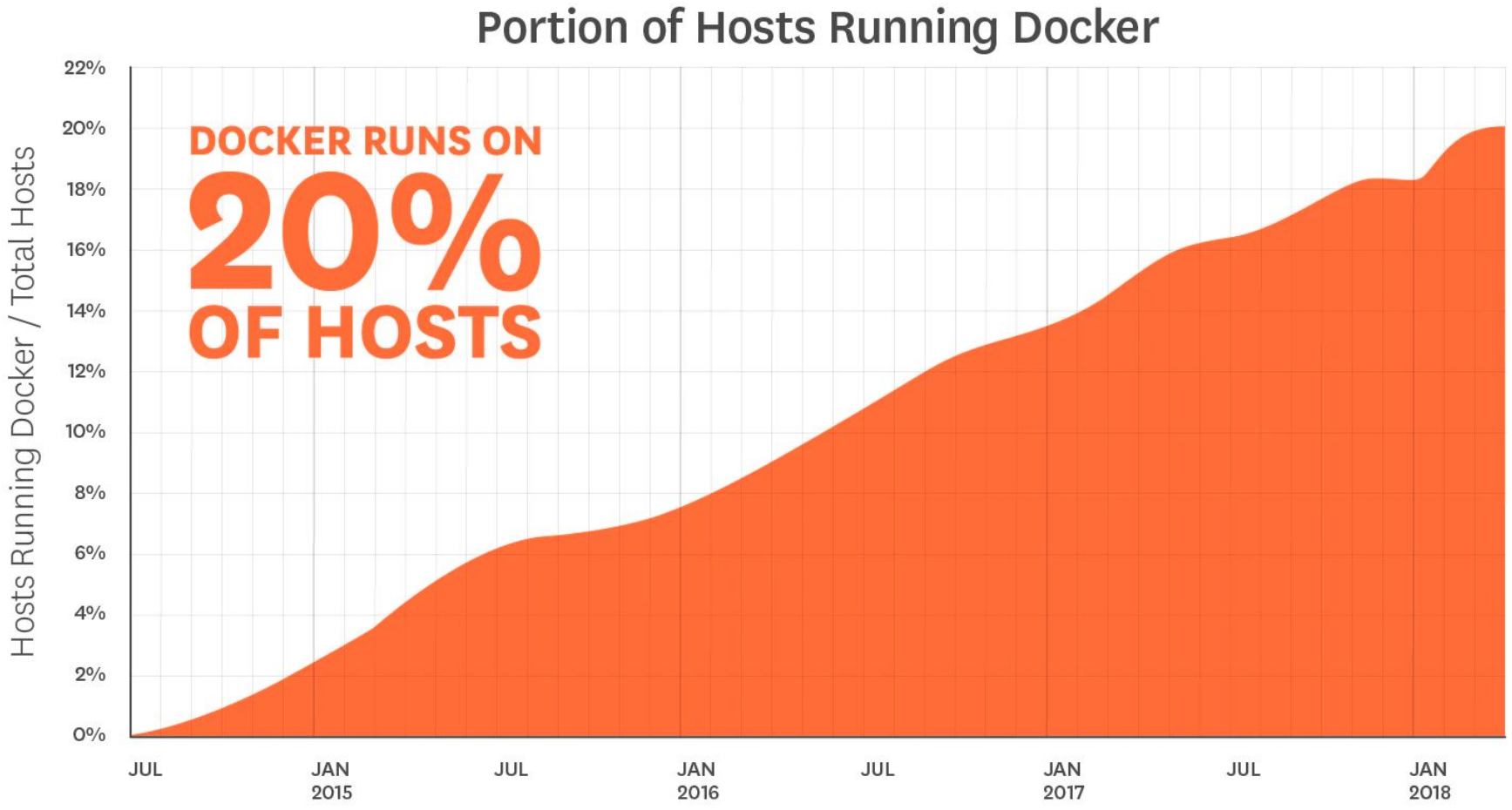
- u mnie działa (a na produkcji nie)
- “potrzebuję biblioteki w wersji 3.243 i ½“
- problem środowiska developerskiego
- ujednolicenie środowisk (linux, Windows)
- uproszczenie CI
- lepsze wykorzystanie zasobów sprzętowych
- szybciej

3.1. Docker

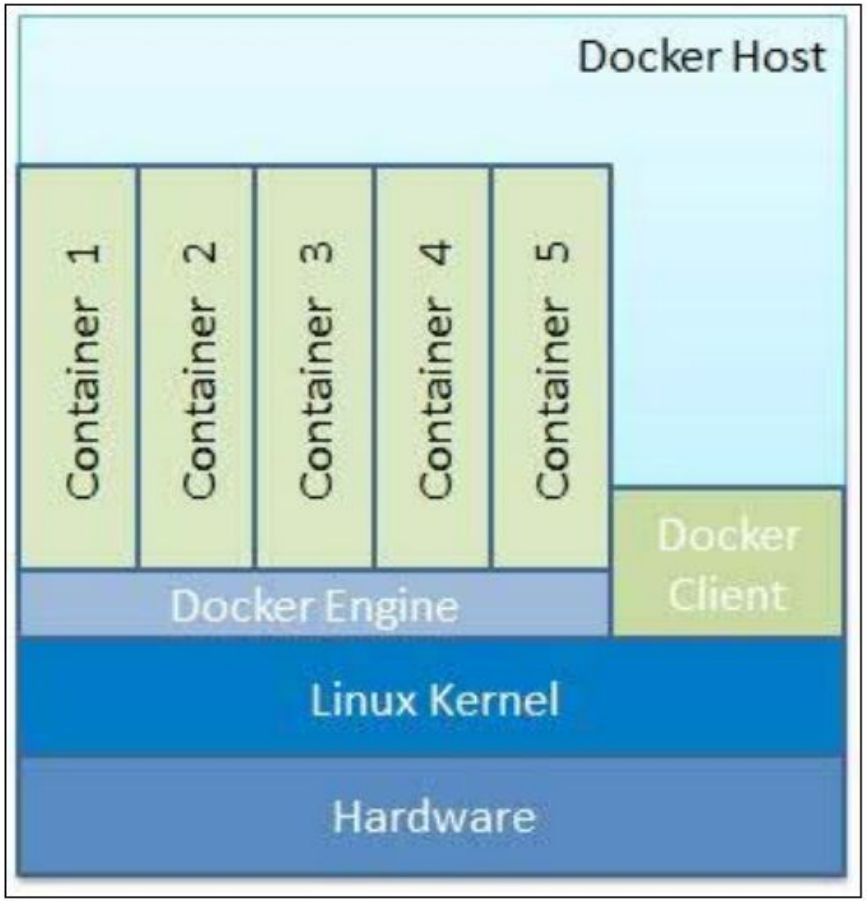
Docker Adoption Behavior

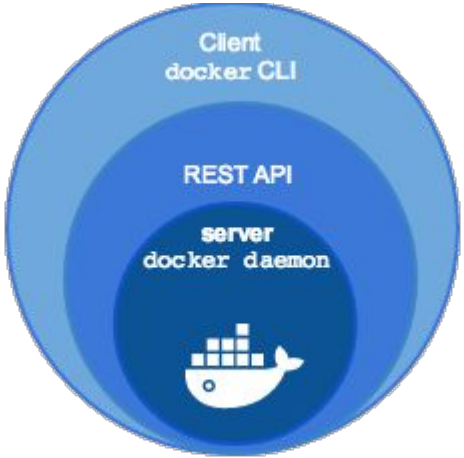


Source: Datadog

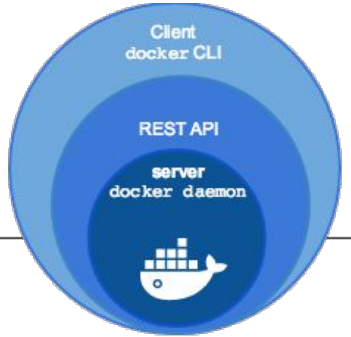


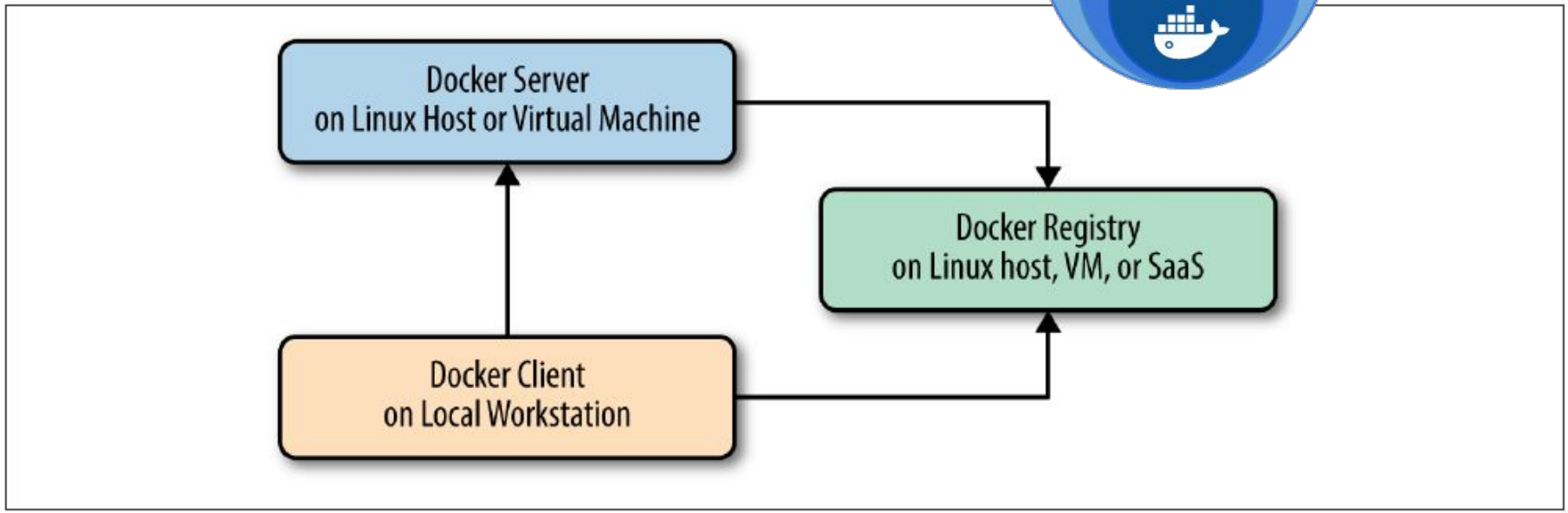
Source: Datadog


Docker Host
Container 1
Container 2
Container 3
Container 4
Container 5
Docker Client
Docker Engine
Linux Kernel
Hardware


Client
docker CLI
REST API
server
docker daemon

Docker client/server





docker server - docker daemon (dockerd)
docker client - command line tool (docker)

Ćwiczenie 3.1. - hello docker

```
## wyświetlamy wersję i info o naszej instalacji dockera
    docker --version
    docker version
    docker info
```

Ćwiczenie 3.1. - hello docker

wyświetlamy wersję i info o naszej instalacji dockera

docker --version

docker version

docker info

odpalamy pierwszy kontener (z obrazu hello-world)

```
> docker run hello-world
```

Ćwiczenie 3.1. - hello docker

wyświetlamy wersję i info o naszej instalacji dockera

```
docker --version
docker version
docker info
```

odpalamy pierwszy kontener (z obrazu hello-world)

```
> docker run hello-world
```

Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world

Hello from Docker!

```
This message shows that your installation appears to be
working correctly.
To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
```

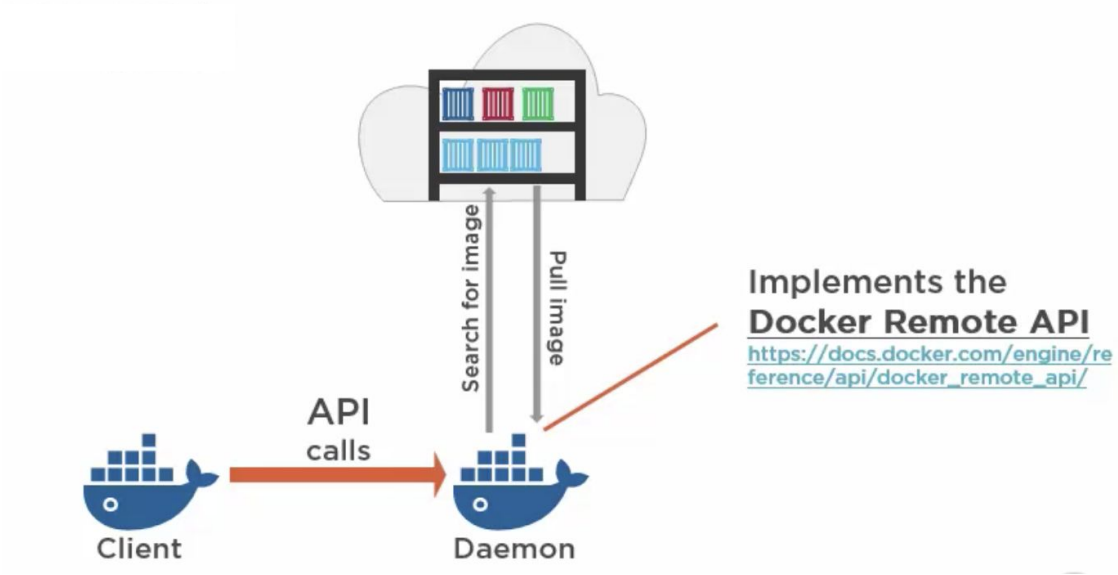
Hello from Docker!

```
This message shows that your installation appears to be
working correctly.
To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from
the Docker Hub.
```

Hello from Docker!

```
This message shows that your installation appears to be
working correctly.
To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from
the Docker Hub.
 3. The Docker daemon created a new container from that
image which runs the executable that produces the output
you are currently reading.
 4. The Docker daemon streamed that output to the Docker
client, which sent it to your terminal.
```

Architektura



Ćwiczenie 3.2. - obrazy dockera

1. (3 min) Wpisz polecenie “docker” - przejrzyj listę komend (sekcja “Commands”) które oferuje docker

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

2. Wyświetl listę obrazów jakie masz pobrane na swoim docker host

Ćwiczenie 3.2.- obrazy dockera

1. (3 min) Wpisz polecenie “docker” - przejrzyj listę komend (sekcja “Commands”) które oferuje docker

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

2. Wyświetl listę obrazów jakie masz pobrane na swoim docker host
3. Usuń obraz “hello-world” ze swojego docker hosta, wylistuj obrazy raz jeszcze, zweryfikuj czy “hello-world” zniknął
4. Pobierz obraz “hello-world” (użyj docker pull !!), ponownie wylistuj obrazy
5. Spróbuj pobrać (pull) obraz jeszcze raz. Jaki komunikat otrzymałeś?
6. Usuń obraz “hello-world”, pobierz i uruchom (2 w 1) kontener z obrazu “hello-world”; (obserwuj pobieranie obrazu na konsoli)

Ćwiczenie 3.2. - obrazy dockera

wyświetlamy wszystkie obrazy zapisane lokalnie

```
docker images
```

usuwamy obraz hello-world

```
docker rmi hello-world
```

usuwamy kontener

```
docker rm <nazwa_kontenera>
```

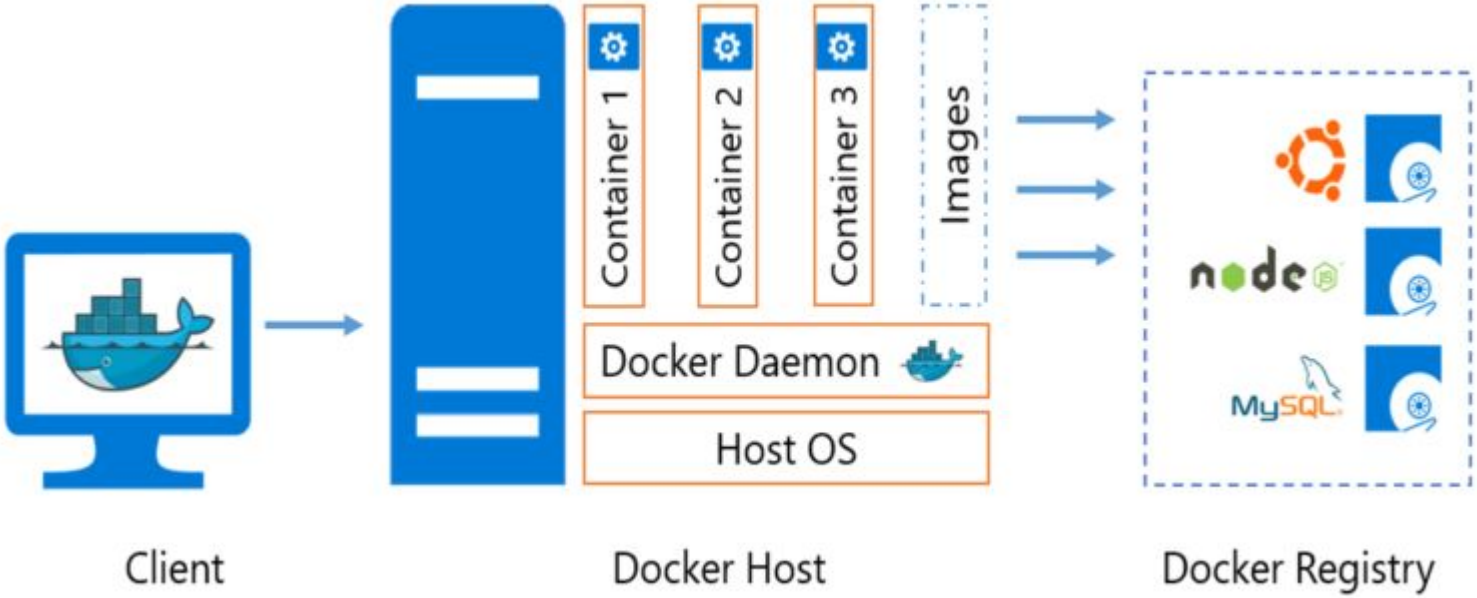
pobieramy obraz (bez uruchamiania kontenera !!)

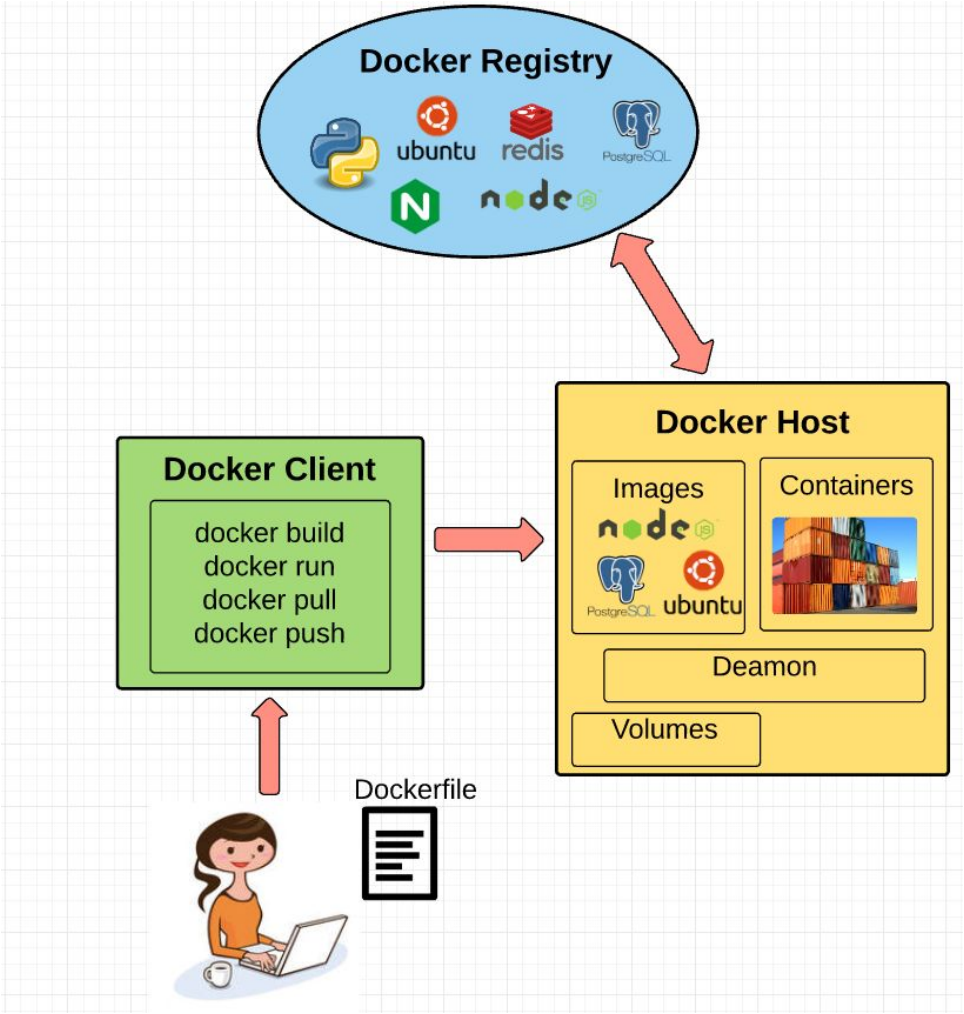
```
docker pull <nazwa_obrazu>
```

Docker

- client
- Docker host
  (engine)
  (daemon)
  (docker server)
- Docker
  registry

3-ech muszkieterów




Docker Registry
ubuntu
redis
PostgreSQL
node
Docker Client
docker build
docker run
docker pull
docker push
Docker Host
Images
node
PostgreSQL
ubuntu
Containers
Deamon
Volumes
Dockerfile

3.1.1 Docker official images

Official repositories - examples

redis

ubuntu

The Official Ubuntu base image

WordPress

WordPress is a free and open source blogging tool and a content management system

MySQL

Popular open-source relational database management system

mongoDB

Document-oriented NoSQL database

CentOS

Official CentOS base image

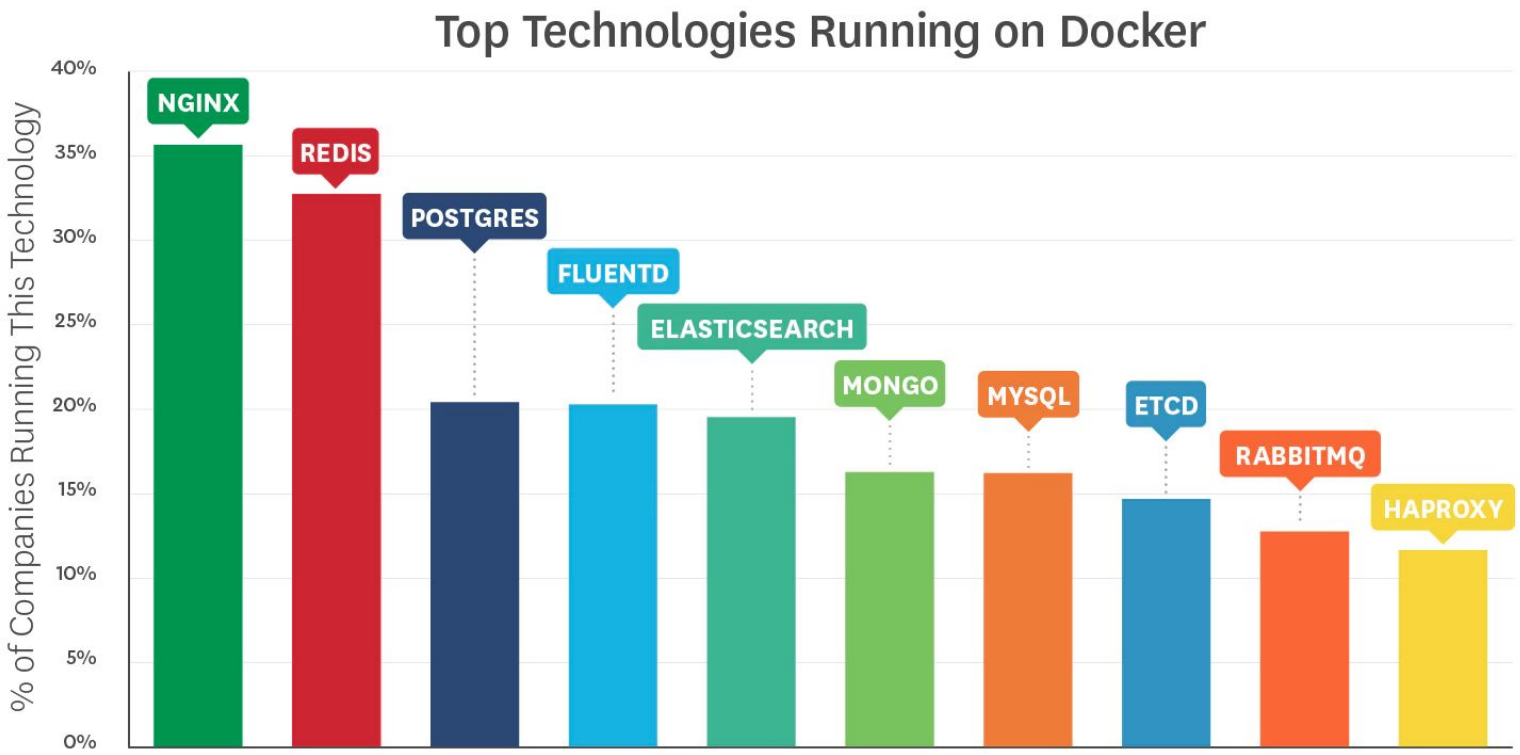
NGINX

High performance reverse proxy server

Relational database management system

node.js

Node.js is a platform for scalable server-side and networking applications



Source: Datadog

Docker hub images

1. standardized base images are well maintained and updated frequently to address security advisories and critical bug fixes

Docker hub images

1. standardized base images are well maintained and updated frequently to address security advisories and critical bug fixes
2. base images are built, validated, and supported by Docker Inc. and are easily recognized by their single word names (e.g.: *centos*).

Docker hub images

1. standardized base images are well maintained and updated frequently to address security advisories and critical bug fixes
2. base images are built, validated, and supported by Docker Inc. and are easily recognized by their single word names (e.g.: *centos*).
3. user members of the Docker community also provide and maintain prebuilt images (pattern: docker_hub_username/image_name e.g.: *tutum/centos*, *picoded/tomcat7*)

Secure | https://hub.docker.com

docker hub

Search

Explore Help Sign in

Docker Hub

Dev-test pipeline automation, 100,000+ free apps, public and private registries

New to Docker?

Create your free Docker ID to get started.

Choose a Docker ID

Email address

Choose a password

- [ ] * I agree to Docker's Terms of Service.
- [ ] * I agree to Docker's Privacy Policy and Data Processing Terms.
- [ ] I would like to receive email updates from Docker, including its various services and products

Sign Up

Ćwiczenie 3.3. - jenkins & docker

1. Używając Docker Huba znajdź obraz jenkinsa.Który obraz jest oficjalny? Jak namierzyć ten najlepszy?

Ćwiczenie 3.3. - jenkins & docker

1. Używając Docker Huba znajdź obraz jenkinsa.Który obraz jest oficjalny? Jak namierzyć ten najlepszy?
2. Zapoznaj się z dokumentacją obrazu jenkins/jenkins (przejrzyj całą sekcję “Usage” dokumentacji), odpal kontener udostępniając tylko port 8080, bez wolumenu (opcja -v). W razie potrzeby wykorzystaj pomoc polecenia run (“*docker run --help*”)
3. Używając “*docker ps*” sprawdź jakie kontenery masz odpalone; jaką nazwę ma uruchomiony kontener z jenkinsem?
4. Połącz się z jenkinsem poprzez przeglądarkę internetową (*localhost:8080*)

odpalamy kontener z podanego obrazu, udostępniając port na zewnątrz

```
docker run -p <hostPort>:<containerPort> jenkins/jenkins
```

sprawdzamy odpalone kontenery

```
docker ps
```

(## lub wszystkie zapisane: docker ps -a)

inspekcja konkretnego kontenera

```
docker inspect <container_name>
```

Ćwiczenie 3.4. - kontenery

1. Usuń istniejący kontener z jenkinsem
2. Odpal dwa nowe kontenery z obrazu jenkins/jenkins, nazwij je odpowiednio: jenkins1, jenkins2; pamiętaj o udostępnieniu portu 8080 dla hosta
3. Zweryfikuj możliwość zalogowania się do każdego z nich poprzez przeglądarkę
4. Zatrzymaj (docker stop) kontener jenkins1. Odpal “docker ps”. Czy widzisz kontener jenkins1?
5. Wznów kontener jenkins1.

Ćwiczenie 3.4. - kontenery - komendy

```
docker ps -a

docker start <containerName>

docker stop <containerName>


docker pause <containerName>

docker unpause <containerName>
```

Kontenery - podsumowanie

1. Obraz (docker image) jest jak klasa w javie

Kontenery - podsumowanie

1. Obraz (docker image) jest jak klasa w javie
2. Kontener (docker container) jest jak obiekt

Kontenery - podsumowanie

1. Obraz (docker image) jest jak klasa w javie
2. Kontener (docker container) jest jak obiekt
3. Na jednym docker hoście mogę mieć kilka kontenerów opartych na tym samym obrazie.

Kontenery - podsumowanie

1. Obraz (docker image) jest jak klasa w javie
2. Kontener (docker container) jest jak obiekt
3. Na jednym docker hoście mogę mieć kilka kontenerów opartych na tym samym obrazie.
4. Kontenery (jak obiekty w javie) mają swój stan i mutują

Kontenery - podsumowanie

1. Obraz (docker image) jest jak klasa w javie
2. Kontener (docker container) jest jak obiekt
3. Na jednym docker hoście mogę mieć kilka kontenerów opartych na tym samym obrazie.
4. Kontenery (jak obiekty w javie) mają swój stan i mutują
5. Zatrzymany kontener można przywrócić do działania bez utraty danych

Skąd obrazy ? - Idź Pan do galerii ..

There are three major locations to store the images you are creating:

1. **Docker Hub**: run by Docker, can contain public and private repositories
2. **Docker Trusted Registry**: (Docker EE) - on-premises or in private virtual cloud; provides the ability to get support from Docker
3. **The locally run Docker registry**: Locally run by yourself to storage images

Przydatne komendy

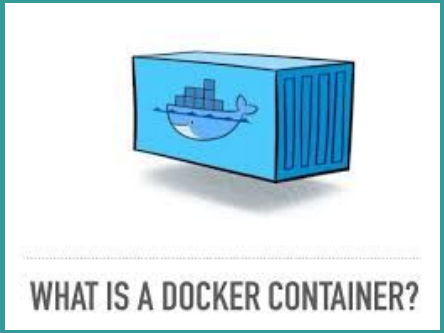
docker search <what>

Przydatne komendy

$ docker search ubuntu

NAME	DESCRIPTION	STARS	OFFICIAL	AUTOMATED
ubuntu	Ubuntu is a Debian-based Linux operating s...	8597	[OK]	
dorowu/ubuntu-desktop-lxde-vnc	Ubuntu with openssh-server and NoVNC	231		[OK]
rastasheep/ubuntu-sshd	Dockerized SSH service, built on top of of...	176		[OK]
consol/ubuntu-xfce-vnc	Ubuntu container with "headless" VNC sessi...	130		[OK]
ansible/ubuntu14.04-ansible	Ubuntu 14.04 LTS with ansible	95		[OK]
ubuntu-upstart	Upstart is an event-based replacement for ...	92	[OK]	
neurodebian	NeuroDebian provides neuroscience research...	54	[OK]	
1and1internet/ubuntu-16-nginx-php-phpmyadmin-mysql-5	ubuntu-16-nginx-php-phpmyadmin-mysql-5	48		[OK]
ubuntu-debootstrap	debootstrap --variant=minbase --components...	40	[OK]	
nuagebec/ubuntu	Simple always updated Ubuntu docker images...	23		[OK]
tutum/ubuntu	Simple Ubuntu docker images with SSH access	18		
i386/ubuntu	Ubuntu is a Debian-based Linux operating s...	14		
1and1internet/ubuntu-16-apache-php-7.0	ubuntu-16-apache-php-7.0	13		[OK]
ppc64le/ubuntu	Ubuntu is a Debian-based Linux operating s...	12		
eclipse/ubuntu_jdk8	Ubuntu, JDK8, Maven 3, git, curl, nmap, mc...	6		[OK]
1and1internet/ubuntu-16-nginx-php-5.6-wordpress-4	ubuntu-16-nginx-php-5.6-wordpress-4	6		[OK]
codenvy/ubuntu_jdk8	Ubuntu, JDK8, Maven 3, git, curl, nmap, mc...	4		[OK]
darksheer/ubuntu	Base Ubuntu Image -- Updated hourly	4		[OK]
pivotaldata/ubuntu	A quick freshening-up of the base Ubuntu d...	2		
1and1internet/ubuntu-16-sshd	ubuntu-16-sshd	1		[OK]
smartentry/ubuntu	ubuntu with smartentry	1		[OK]
ossobv/ubuntu	Custom ubuntu image from scratch (based on...	0		
paasmule/bosh-tools-ubuntu	Ubuntu based bosh-cli	0		[OK]
1and1internet/ubuntu-16-healthcheck	ubuntu-16-healthcheck			[OK]
pivotaldata/ubuntu-gpdb-dev	Ubuntu images for GPDB development			

3.1.2. Kontenery z bliska



Start kontenera

docker run

=

docker create + docker start

Start kontenera

```
$ docker run --rm -ti debian:latest /bin/bash
```

--rm	po skończonej robocie usuń kontener
-t	allocate a psuedo-TTY
-i	interactive session - keep STDIN open
-d, --detach	run container in background and print container ID

Ćwiczenie 3.5. - kontenery

1. *docker run --name mongo1 mongo*
2. zatrzymaj kontener mongo1 (np. Ctrl+C) i wystartuj kolejny:

   docker run --name mongo2 -d mongo

   który kontener działa (docker ps)?

3. *docker run --name debian1 debian*
4. *docker run --name debian2 -it debian bash*

Ćwiczenie 3.5. - kontenery

1. *docker run --name mongo1 mongo*
2. zatrzymaj kontener mongo1 (np. Ctrl+C) i wystartuj kolejny:

   docker run --name mongo2 -d mongo

   który kontener działa (docker ps)?

3. *docker run --name debian1 debian*
4. *docker run --name debian2 -it debian bash*

WNIOSEK:

Niektóre kontenery zaraz po starcie “plują” na konsolę, a inne nie.

ściąga

`docker run`	Starts a new container
`docker pull`	Copies images to the Docker Host
`docker images`	Lists images on the Docker Host
`docker rmi`	Removes images from the Docker Host
`docker ps`	Lists running containers
`docker stop`	Stops running containers
`docker rm`	Removes (**deletes**) stopped containers

Inne komendy

delete all of the containers on your Docker hosts :

```
$ docker rm $(docker ps -a -q)
```

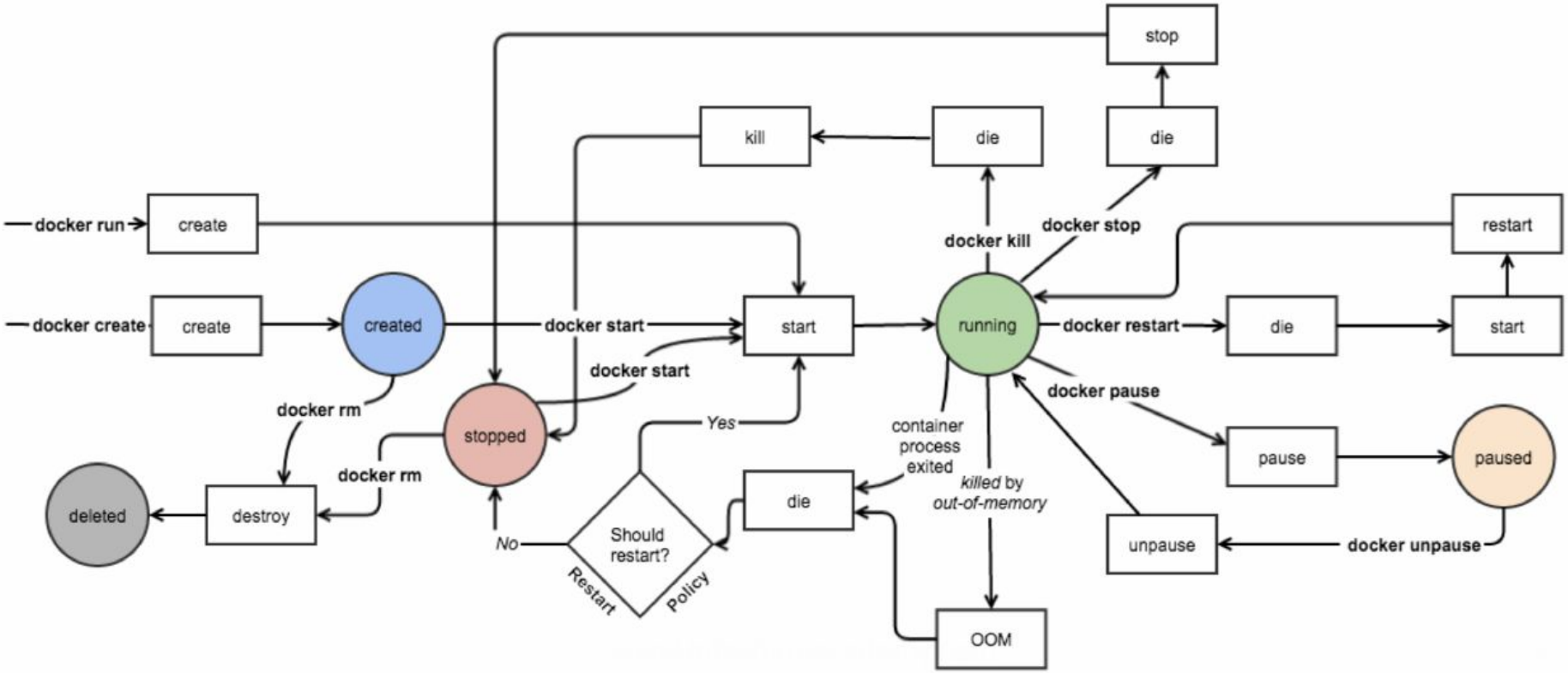
remove all untagged images:

```
$ docker rmi $(docker images -q -f "dangling=true")
```

delete all the images on your Docker host:

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

Container FSM



3.1.3. Docker networking

bez łączności nie ma nic

Po co nam sieci w dockerze?

- kontenery muszą się łączyć ze światem zewnętrznym

Po co nam sieci w dockerze?

- kontenery muszą się łączyć ze światem zewnętrznym
- świat zewnętrzny musi się łączyć z kontenerami świadczącymi usługi (np. kontener z jenkinsem, kontener bazodanowy itp.)

Po co nam sieci w dockerze?

- kontenery muszą się łączyć ze światem zewnętrznym
- świat zewnętrzny musi się łączyć z kontenerami świadczącymi usługi (np. kontener z jenkinsem, kontener bazodanowy itp.)
- komunikacja kontener - kontener

Network drivers in Docker

- none
- **host**

  no network isolation between the container and the Docker host

  host network is used directly by container

Network drivers in Docker

- none
- **host**

  no network isolation between the container and the Docker host

  host network is used directly by container
- **bridge (default)**

  provides isolation between docker host and container

  forwards traffic between network segments using software bridge

Network drivers in Docker

- none
- **host**

  no network isolation between the container and the Docker host

  host network is used directly by container
- **bridge (default)**

  provides isolation between docker host and container

  forwards traffic between network segments using software bridge
- overlay
- macvlan

Ćwiczenie 3.6. - host network

Odpal dwa kontenery używając poleceń:

- docker run **--network host** --name nginx1 -d nginx
- docker run **--network host** --name nginx2 -d nginx

- zweryfikuj działanie serwera (przeglądarka lub: *curl localhost:80)*
- sprawdź wszystkie aktywne kontenery, co się stało?
- używając polecenia ***docker logs*** sprawdź logi dla obu kontenerów
- (*) powtórz ćwiczenie bez użycia przełącznika -d

Bridge network driver in Docker

Default “bridge”

- NAME = “bridge”
- always exists
- use as default

User-defined bridge

- NAME = “<defined by user>”
- need to be manually created/removed
- *docker network create <name>*

Bridge network driver in Docker

Default “bridge”

- NAME = “bridge”
- always exists
- use as default

User-defined bridge

- NAME = “<defined by user>”
- need to be manually created/removed
- *docker network create <name>*

Kiedy nie zdefiniujemy do jakiej sieci ma zostać podłączony nasz kontener - docker engine użyje sieci “bridge”

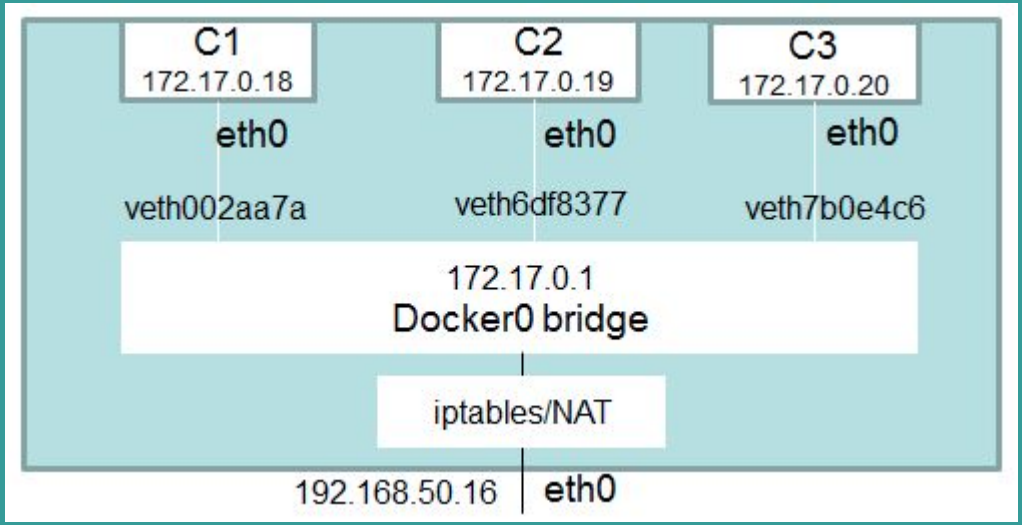
Ćwiczenie 3.7. - “bridge” network

Odpal dwa kontenery używając poleceń:

- docker run **-p 8080:80** --name nginx1 -d nginx
- docker run **-p 8080:80** --name nginx2 -d nginx

- czy widzisz błąd w poleceniu? - popraw
- zweryfikuj działanie obu (!!) serwerów, sprawdź ich logi
- ***docker network ls*** - jakie sieci widzisz?
- dokonaj inspekcji sieci o nazwie “bridge” (**docker network inspect**) - jakie kontenery są do niej podłączone?

“bridge” (default) network



wydaj polecenie na hoście: *ifconfig docker0*

http://106.51.226.114:9191/lab-7-docker-networking/

```
$ docker network --help
Usage:  docker network COMMAND

Manage networks

Options:
      --help   Print usage

Commands:
  connect     Connect a container to a network
  create      Create a network
  disconnect  Disconnect a container from a network
  inspect     Display detailed information on one or more networks
  ls          List networks
  prune       Remove all unused networks
  rm          Remove one or more networks

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

$ docker port --help

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

List port mappings or a specific mapping for the container

Options:

--help Print usage

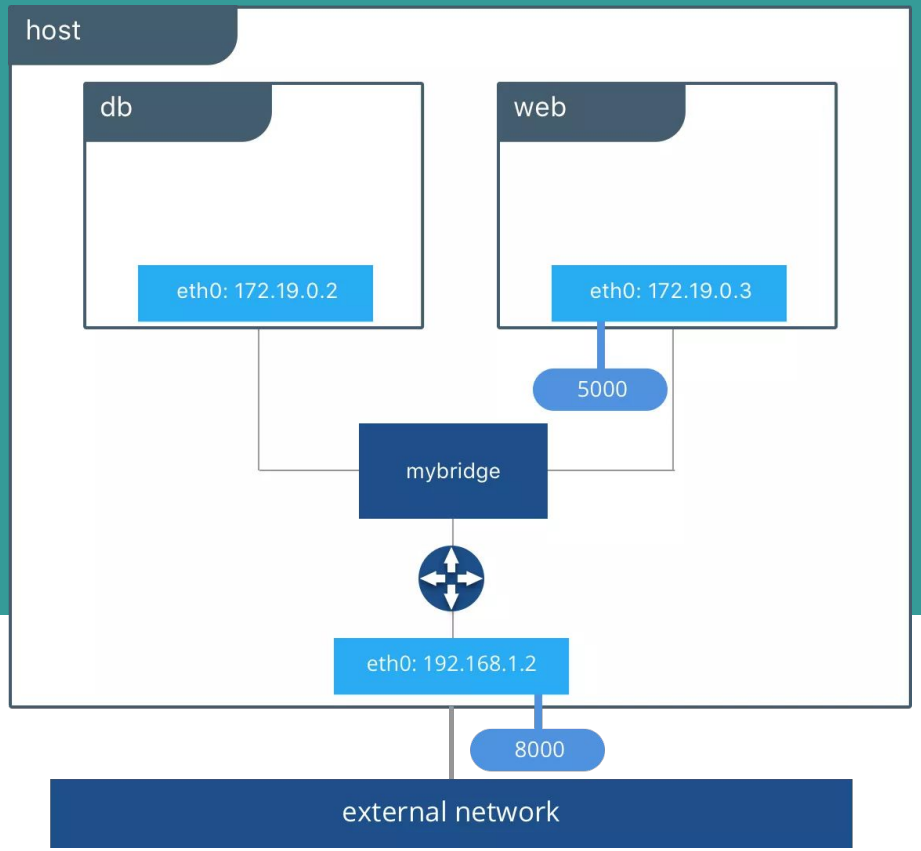
Ćwiczenie 3.8. - user-defined bridge network

Odpal dwa kontenery używając poleceń:

- docker run **--network net1** -it --name debian1 debian
- docker run **--network net1** -it --name debian2 debian

1. ***docker network ls*** - jakie sieci widzisz?
2. dokonaj inspekcji sieci o nazwie “net1” (**docker network inspect**) - jakie kontenery są do niej podłączone?, jakie IP mają, jaki jest default gateway?
3. zweryfikuj IP kontenerów (polecenie “*ip address*” w kontenerze)?
4. dokonaj pingowania z kontenera do kontenera (w obie strony)
5. zamiast adresów IP do pingowania użyj nazw kontenerów
6. czy oba stworzone kontenery mają dojście do internetu? - zweryfikuj to

user-defined bridge network



docker networks

- Docker daemon effectively acts as a DHCP server for each container
- na starcie kontener jest podłączony do jednej sieci (default = “bridge”)

3.1.4. Docker - wolumeny

jak zapisywać dane żeby nie zginęły?

Ćwiczenie 3.9. - kontenery i dane

1. Utwórz nowy kontener z debianem:

```
docker run -it --name debian1 debian
```

2. Z poziomu kontenera utwórz nowy plik tekstowy (użyj polecenia echo)
3. zatrzymaj kontener (*docker stop*), potwierdź zatrzymanie *docker ps*
4. wznów kontener (*docker start*), zaloguj się do niego (*docker exec*) i dodaj jeszcze jeden plik tekstowy
5. czy utworzony wcześniej plik nadal istnieje?
6. Co zwraca komenda: ***docker diff debian1*** ?

$ **docker exec** --help

Usage: **docker exec [OPTIONS] CONTAINER COMMAND [ARG...]**

Run a command in a running container

Options:

-d, --detach	Detached mode: run command in the background
--help	Print usage
-i, --interactive	Keep STDIN open even if not attached
-t, --tty	Allocate a pseudo-TTY
-u, --user string	Username or UID (format: <name|uid>[:<group|gid>])

Przykład:

$ docker exec -it debian1 bash

$ docker exec nginx "yum –y update nginx"

$ **docker diff** --help

Usage: **docker diff CONTAINER**

Inspect changes to files or directories on a container's filesystem

Options:
--help Print usage

Przykład:

$ docker diff debian1

output:

```
C /root
A /root/.bash_history
A /myfile.txt
```

A - added
C - changed

Volumes in Docker

- Data volumes are designed to persist data, independent of the container’s life cycle.

Volumes in Docker

- Data volumes are designed to persist data, independent of the container's life cycle.
- Data volumes persist even if the container itself is deleted

Volumes in Docker

- Data volumes are designed to persist data, independent of the container's life cycle.
- Data volumes persist even if the container itself is deleted
- Volumes are initialized when a container is created.

Volumes in Docker

- Data volumes are designed to persist data, independent of the container's life cycle.
- Data volumes persist even if the container itself is deleted
- Volumes are initialized when a container is created.
- Data volumes can be shared and reused among containers.

Volumes in Docker

- Data volumes are designed to persist data, independent of the container's life cycle.
- Data volumes persist even if the container itself is deleted
- Volumes are initialized when a container is created.
- Data volumes can be shared and reused among containers.
- Docker never automatically deletes volumes when user remove a container, nor will it "garbage collect" volumes that are no longer referenced by a container.

Volumes in Docker

- **Volumes** are stored in a part of the host filesystem which is *managed by Docker* (`/var/lib/docker/volumes/` on Linux).

Volumes in Docker

- **Volumes** are stored in a part of the host filesystem which is *managed by Docker* (`/var/lib/docker/volumes/` on Linux).
- When you mount the volume into a container, this directory is what is mounted into the container

named / anonymous volumes

- When you mount a volume, it may be **named** or **anonymous**.
- Anonymous volumes are not given an explicit name when they are first mounted into a container, so Docker gives them a random name that is guaranteed to be unique within a given Docker host.
- Besides the name, named and anonymous volumes behave in the same ways.

Ćwiczenie 3.10. - wolumeny

1. utwórz nowy kontener nginx:

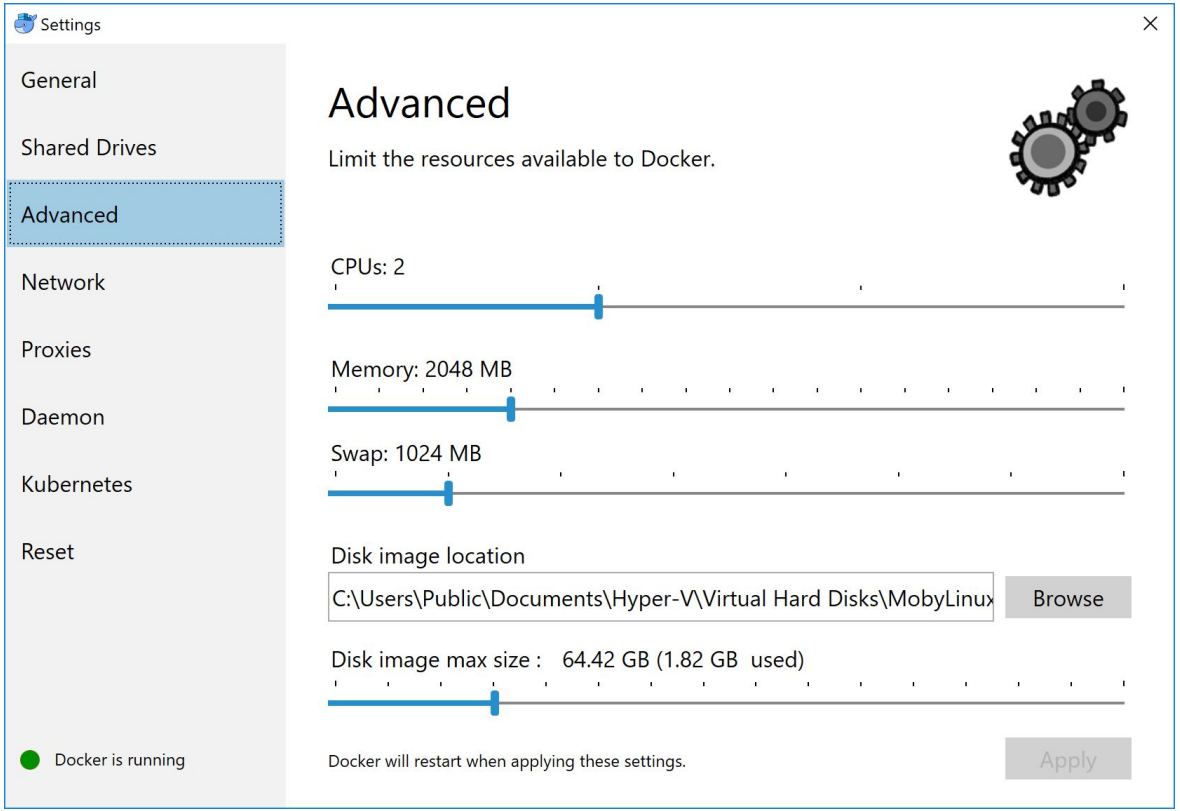
```
$ docker run -d -p 81:80 --name=nginxtest \
  -v nginx-vol:/usr/share/nginx/html  \
  nginx
```

2. sprawdź info o wolumenie (*docker inspect nginxtest*)
3. z poziomu hosta zajrzyj do folderu gdzie mieści się wolumen, zmień plik index.html (np. dodaj swoje imię). Sprawdź zmiany w przeglądarce www.
4. usuń kontener nginxtest
5. wystartuj nowy kontener nginx o nazwie “nginxtest2”, użyj tego samego volumenu, zweryfikuj działanie w przeglądarce www
6. (*)wystartuj kolejny kontener (“nginxtest3”) używający tego samego wolumenu, zweryfikuj działanie obu z nich w przeglądarce

3.1.5. Docker && Microsoft

czy linuxowe kontenery można odpalić na Windzie?

Docker on Windows - MobyLinux



Windows containers

Windows i kontenery

1. hyper-V isolation
   a. mobyLinux + docker
   b. Windows Nano Server

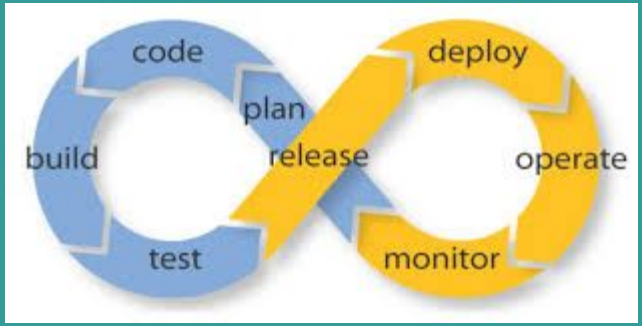
   you can run different windows kernel (e.g Windows 10 (host) use Hyper-V isolation to utilize the Windows Server kernel version and configuration.

2. windows-server containers

- “docker-style” containers on Windows (Microsoft provided the Windows kernel with some of the same mechanisms used in Linux to perform the isolation)
- the same kernel being used - limited isolation and security

3.1.6. Docker, CI, CD

Let’s start with the first set of slides



Advantages of Docker containers

1. **Rapid application deployment**: with minimal runtime, containers can be deployed quickly because of the reduced size (only the application is packaged)

Advantages of Docker containers

1. **Rapid application deployment**: with minimal runtime, containers can be deployed quickly because of the reduced size (only the application is packaged)
2. **Portability**: An application with its operating environment can be bundled together into a single Docker container that is independent from the OS version or deployment model. The Docker containers can be easily transferred to another machine that runs Docker container and executed without any compatibility issues.

   You know that if they start up a container using the Dockerfile, the container will act the same in your environment as it will on others.

Advantages of Docker containers

1. **Rapid application deployment**: with minimal runtime, containers can be deployed quickly because of the reduced size (only the application is packaged)
2. **Portability**: An application with its operating environment can be bundled together into a single Docker container that is independent from the OS version or deployment model. The Docker containers can be easily transferred to another machine that runs Docker container and executed without any compatibility issues.

   You know that if they start up a container using the Dockerfile, the container will act the same in your environment as it will on others.
3. **Easily Shareable**: Pre-built container images can be easily shared with the help of public repositories as well as hosted private repositories for internal use.

Advantages of Docker containers

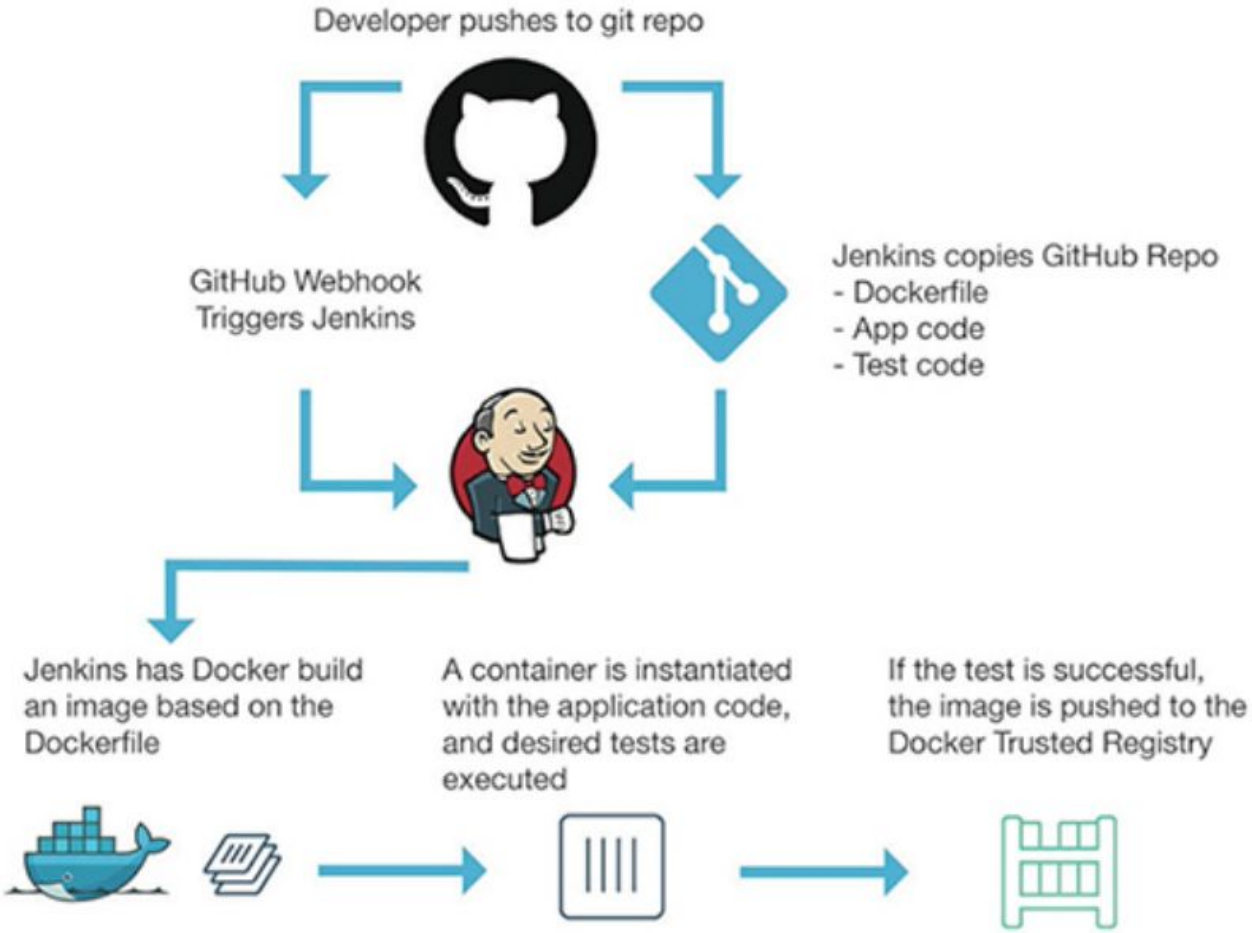
1. **Rapid application deployment**: with minimal runtime, containers can be deployed quickly because of the reduced size (only the application is packaged)
2. **Portability**: An application with its operating environment can be bundled together into a single Docker container that is independent from the OS version or deployment model. The Docker containers can be easily transferred to another machine that runs Docker container and executed without any compatibility issues.

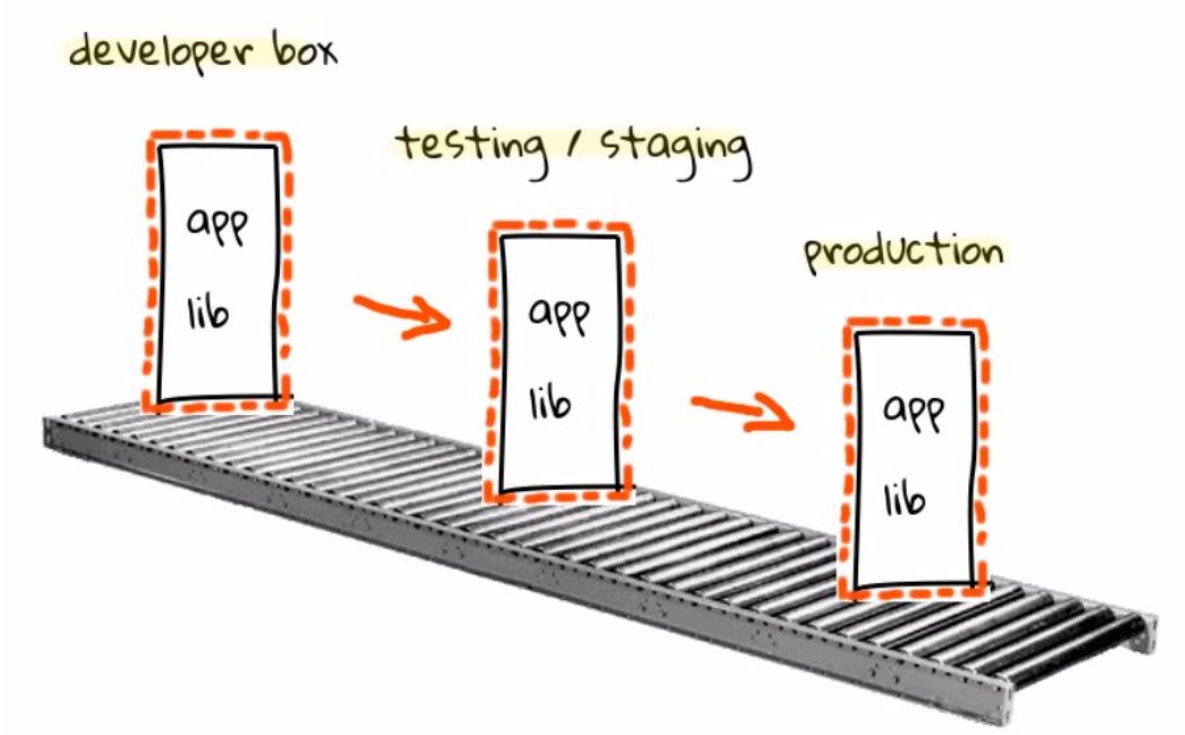
   You know that if they start up a container using the Dockerfile, the container will act the same in your environment as it will on others.
3. **Easily Shareable**: Pre-built container images can be easily shared with the help of public repositories as well as hosted private repositories for internal use.
4. **Lightweight footprint**: Even the Docker images are very small and have a minimal footprint to deploy a new application with the help of containers.

Quick deployment / teardown

With a single command, you can spin up new containers or tear down existing ones.

Typically, if you try to clone a virtual machine or spin up a new one, you are looking at waiting for close to or over a few hours. With Docker, it will take a few minutes to achieve what you need.

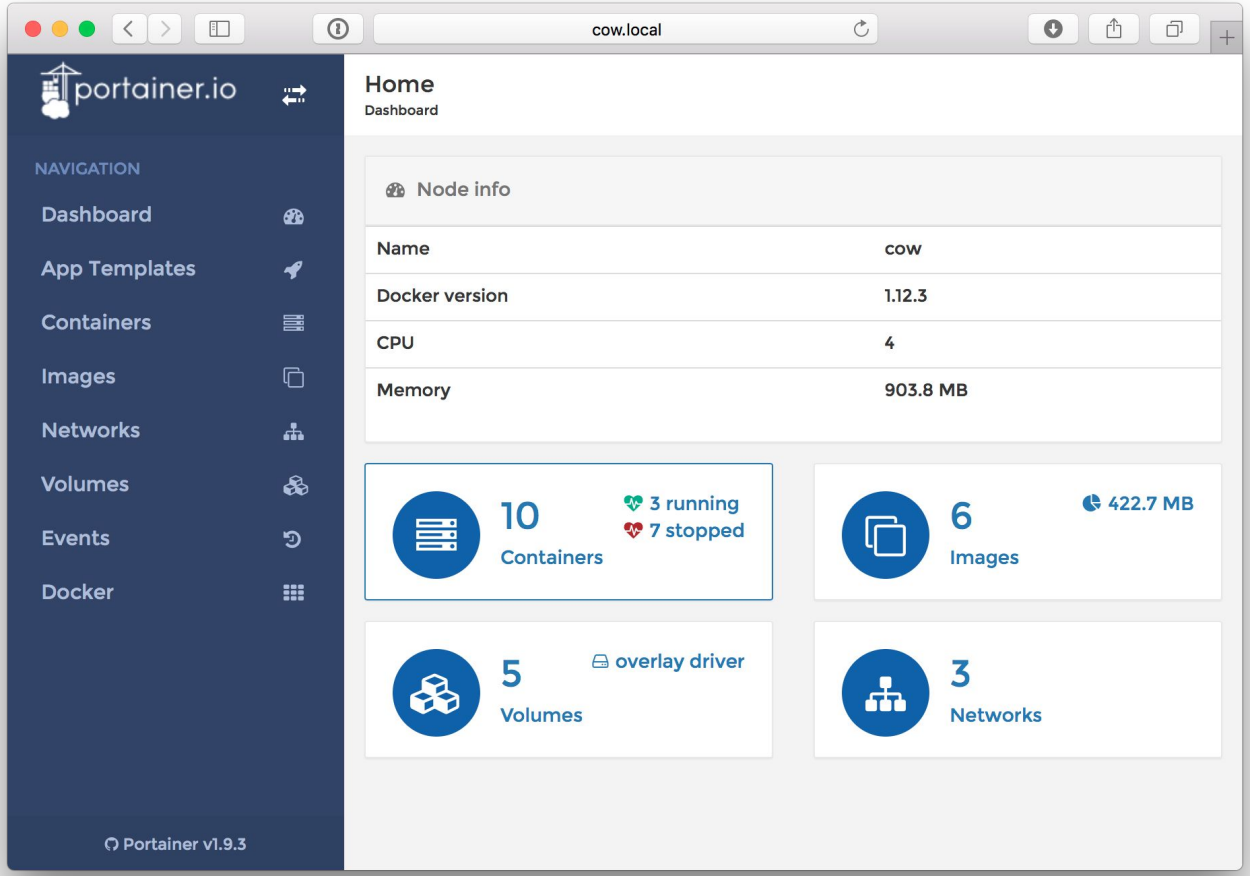

Developer pushes to git repo
GitHub Webhook
Triggers Jenkins
Jenkins copies GitHub Repo
- Dockerfile
- App code
- Test code
Jenkins has Docker build
an image based on the
Dockerfile
A container is instantiated
with the application code,
and desired tests are
executed
If the test is successful,
the image is pushed to the
Docker Trusted Registry


developer box
app
lib
testing / staging
app
lib
production
app
lib

Docker “UI” in the webbrowser

https://hub.docker.com/r/portainer/portainer/

http://localhost:9000/


cow.local
portainer.io
NAVIGATION
Dashboard
App Templates
Containers
Images
Networks
Volumes
Events
Docker
Portainer v1.9.3
Home
Dashboard
Node info
Name
cow
Docker version
1.12.3
CPU
4
Memory
903.8 MB
10
Containers
3 running
7 stopped
6
Images
422.7 MB
5
Volumes
overlay driver
3
Networks

3.1.7. Docker - własny obraz

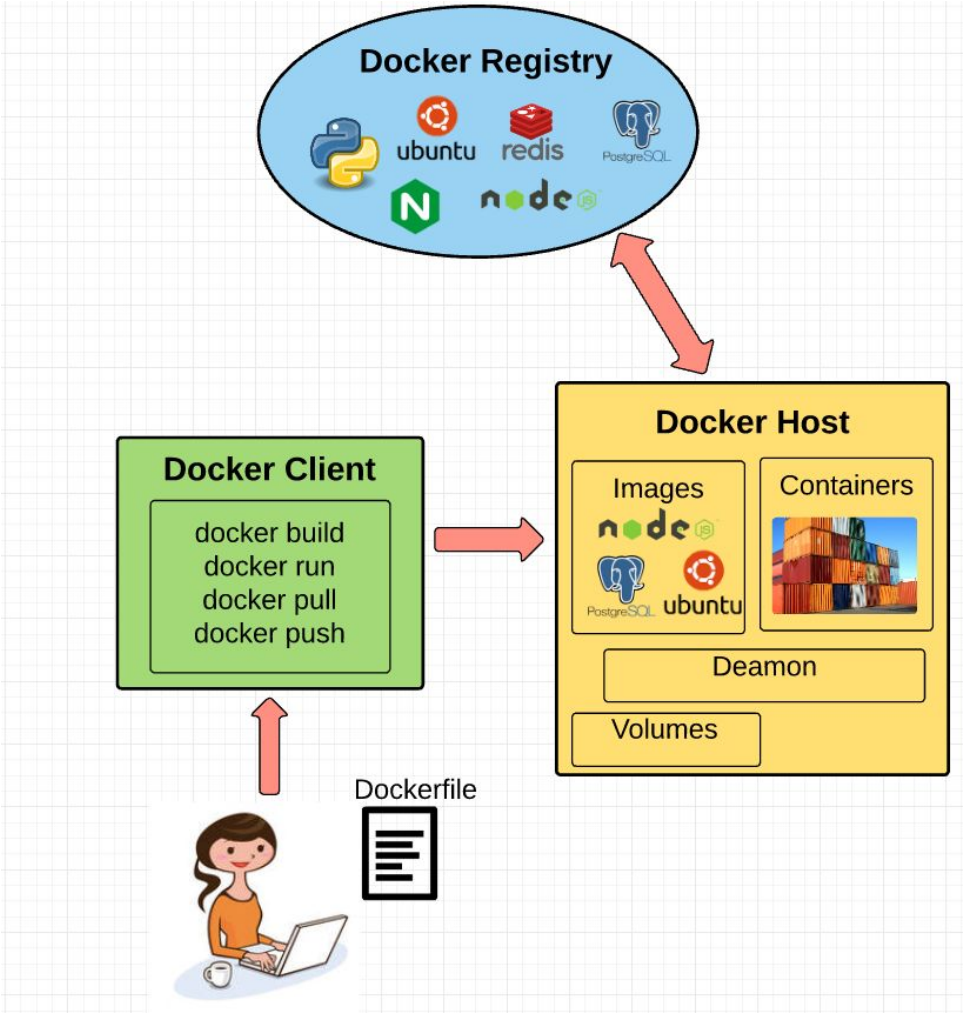
po co się ograniczać, zróbmy coś swojego

Sposoby tworzenia “szytych na miarę” obrazów

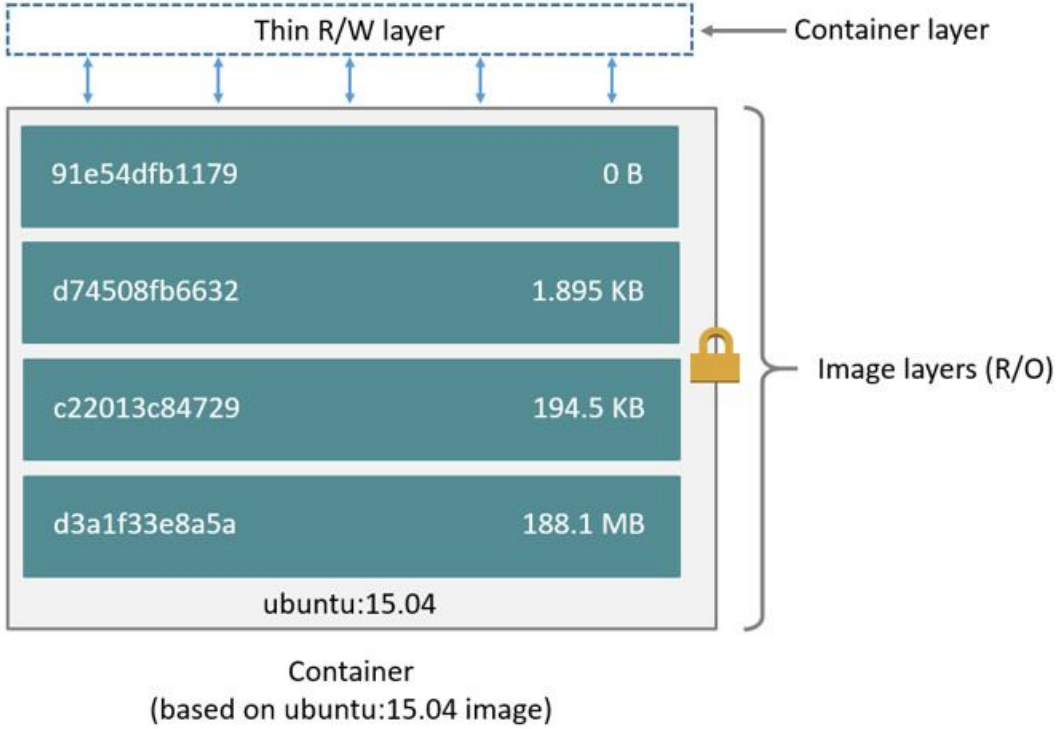
1. docker commit
   a. ściągamy obraz np. Ubuntu, odpalamy kontener
   b. *docker exec -it my_ubuntu bash*
   c. *apt-get install git*
   d. docker commit my_ubuntu grzesiowski/changed_ubuntu:1.0

Sposoby tworzenia “szytych na miarę” obrazów

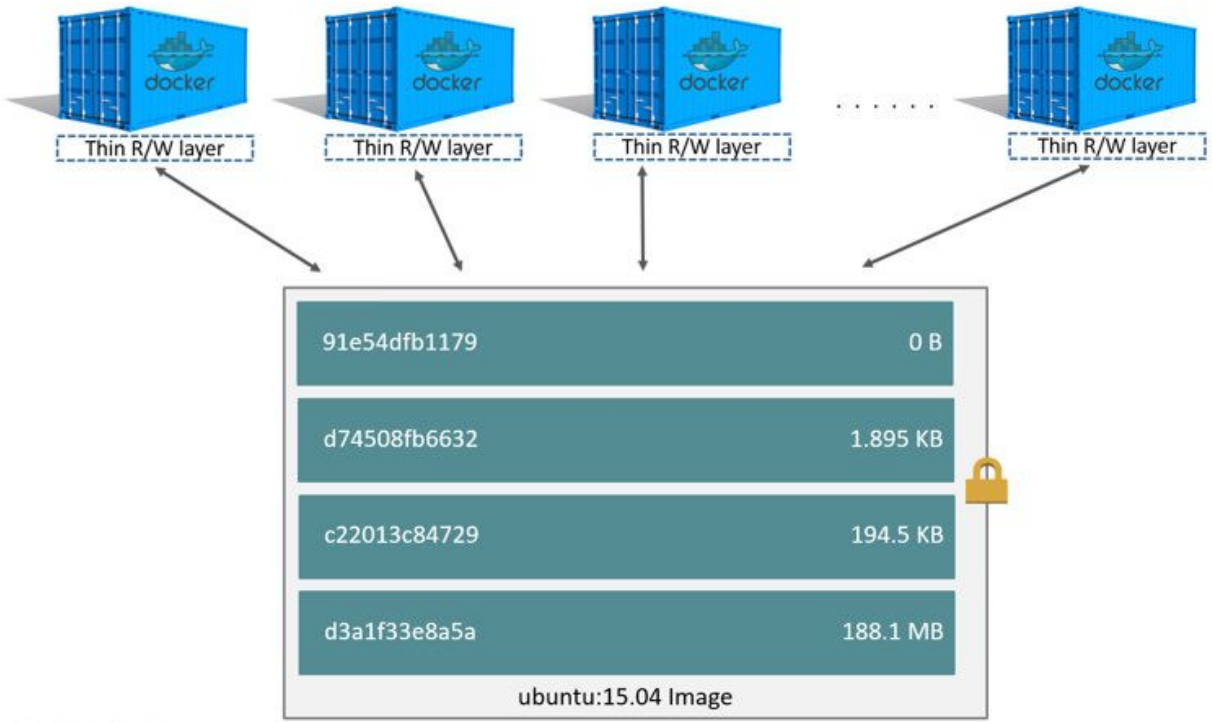
1. docker commit - nie zalecane !!
   a. ściągamy obraz np. Ubuntu, odpalamy kontener
   b. docker exec -it my_ubuntu bash
   c. apt-get install git
   d. docker commit my_ubuntu grzesiowski/changed_ubuntu:1.0
2. Dockerfile - właściwa droga :-)
   a. tworzymy plik “Dockerfile”
   b. definiujemy, że nasz obraz będzie bazować na ubuntu, dodajemy kroki (np. Instalacja git-a)
   c. docker build


Docker Registry
ubuntu
redis
PostgreSQL
node
Docker Client
docker build
docker run
docker pull
docker push
Docker Host
Images
node
PostgreSQL
ubuntu
Containers
Deamon
Volumes
Dockerfile

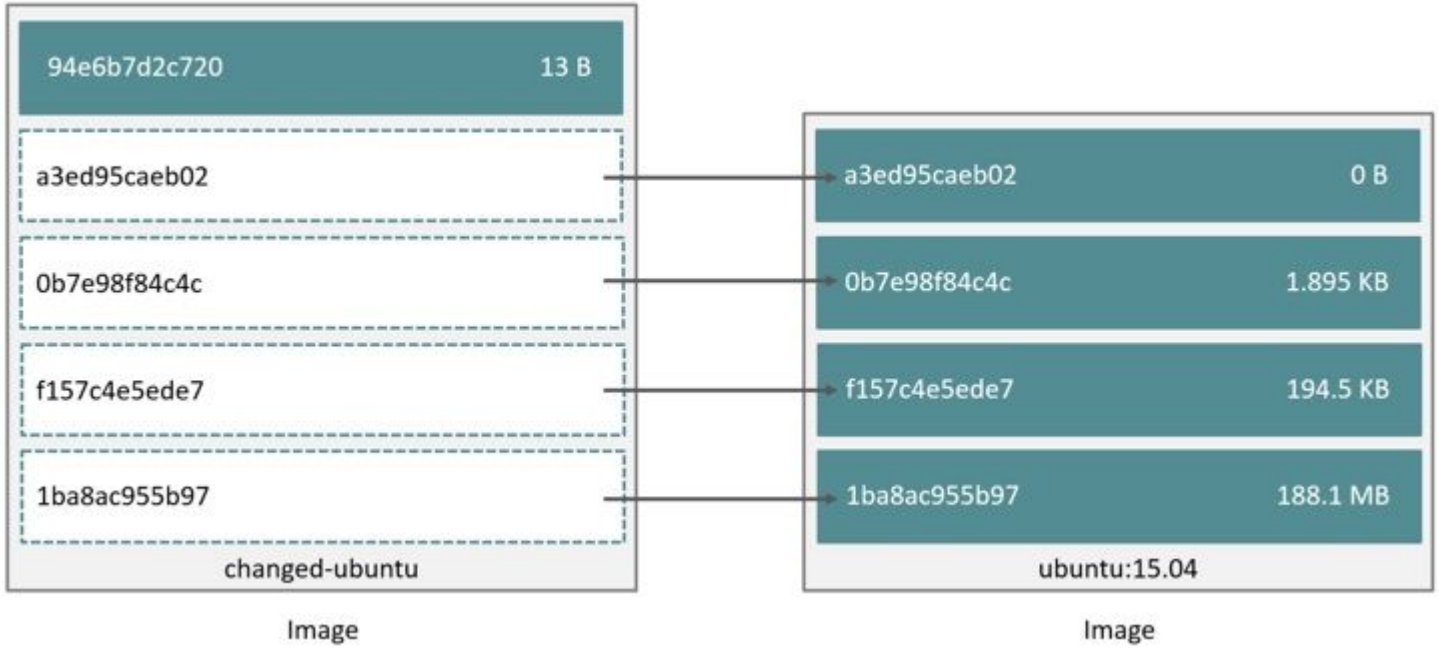
Obraz jest warstwowy



one image, multiple containers



każda zmiana w kontenerze to nowa warstwa



Dockerfile - examples

1. Ubuntu with added “ping” command
https://hub.docker.com/r/adiazmor/docker-ubuntu-with-ping/~/dockerfile/

2. tomcat
https://hub.docker.com/r/davidcaste/alpine-tomcat/~/dockerfile/

3. Mongo db
https://github.com/docker-library/mongo/blob/974dbf4a5f951f4bc627dc59761dab19585edcc4/3.2/Dockerfile

Dlaczego obraz mongo “pluje” na konsole zaraz po starcie?

1. sprawdź Dockerfile dla mongo
2. sprawdź Dockerfile dla ubuntu
3. porównaj ostatnie linie

Dlaczego obraz mongo “pluje” na konsole zaraz po starcie?

DEBIAN Dockerfile

```
FROM scratch
ADD rootfs.tar.xz /
CMD ["bash"]
```

MONGO Dockerfile

```
FROM debian:jessie-slim

...

VOLUME /data/db /data/configdb

COPY docker-entrypoint.sh /usr/local/bin/
RUN ln -s usr/local/bin/docker-entrypoint.sh
/entrypoint.sh
ENTRYPOINT ["docker-entrypoint.sh"]

EXPOSE 27017
CMD ["mongod"]
```

Dockerfile - podstawy

1. **Podstawowy format**

```
# Comment
INSTRUCTION arguments

    ## INSTRUCTION - duże litery (konwencja)
```

2. **Każdy Dockerfile zaczynamy od FROM:**

```
FROM ubuntu

FROM jenkins/jenkins:lts

FROM scratch
```

Parent image vs base images

- A parent image is the image that your image is based on. It refers to the contents of the `FROM` directive in the Dockerfile.
- Each subsequent declaration in the Dockerfile modifies this parent image. (warstwy)
- Most Dockerfiles start from a parent image, rather than a base image. However, the terms are sometimes used interchangeably.

- A base image either has no `FROM` line in its Dockerfile, or has `FROM scratch`.

ćwiczenie 3.11. - pierwszy Dockerfile

1. stwórz nowy plik Dockerfile w nowym(pustym) folderze, (np. *~/docker/cwiczenia/dockerfile/1/Dockerfile*)
2. wklej zawartość do pliku:

```
FROM debian
CMD ["echo", "hello infoShare!"]
```

3. z poziomu folderu “skompiluj” obraz :

```
docker build --tag hello-isa .
```

   Ile warstw zostało stworzonych?

4. uruchom kontener, zweryfikuj działanie
5. czy kontener nadal jest uruchomiony?
6. (*)jaki rozmiar ma obraz hello-isa?
7. uruchom nowy kontener kontener w trybie detached. Czy kontener nadal działa?

ćwiczenie 3.12. - drugi Dockerfile

1. zmodyfikuj Dockerfile:

```
FROM debian

CMD ["ping", "www.infoshareacademy.com", "-c", "100"]
```

2. zbuduj obraz (hello-isa), uruchom kontener, zweryfikuj działanie
3. odpal obraz w trybie detached

CMD

The `CMD` instruction has three forms:

- `CMD ["executable","param1","param2"]` (*exec* form, this is the preferred form)
- `CMD ["param1","param2"]` (as *default parameters to ENTRYPOINT*)
- `CMD command param1 param2` (*shell* form)

There can only be one `CMD` instruction in a `Dockerfile`. If you list more than one `CMD` then only the last `CMD` will take effect.

The main purpose of a `CMD` is to provide defaults for an executing container. These defaults can include an executable, or they can omit the executable, in which case you must specify an `ENTRYPOINT` instruction as well.

> **Note**: If `CMD` is used to provide default arguments for the `ENTRYPOINT` instruction, both the `CMD` and `ENTRYPOINT` instructions should be specified with the JSON array format.

> **Note**: The *exec* form is parsed as a JSON array, which means that you must use double-quotes (") around words not single-quotes (').

> **Note**: Unlike the *shell* form, the *exec* form does not invoke a command shell. This means that normal shell processing does not happen. For example, `CMD [ "echo", "$HOME" ]` will not do variable substitution on `$HOME`. If you want shell processing then either use the *shell* form or execute a shell directly, for example: `CMD [ "sh", "-c", "echo $HOME" ]`. When using the exec form and executing a shell directly, as in the case for the shell form, it is the shell that is doing the environment variable expansion, not docker.

https://docs.docker.com/engine/reference/builder/#cmd

ćwiczenie 3.13.- Dockerfile i “RUN”

1. stwórz nowy, pusty folder z plikiem Dockerfile:

```
FROM debian
RUN echo "line1">>test.txt
RUN echo "line2">>test.txt
CMD ["cat", "test.txt"]
```

2. zbuduj obraz (test-isa), uruchom kontener, zweryfikuj działanie
3. nie wprowadzając zmian w pliku, zbuduj obraz jeszcze raz, porównaj komunikaty
4. zmodyfikuj Dockerfile - dodaj trzecią linię “line3”, porównaj komunikat

```
$ docker build --tag test-isa .
Sending build context to Docker daemon  2.048kB
Step 1/4 : FROM debian
 ---> be2868bebaba
Step 2/4 : RUN echo "line1">>test.txt
 ---> Running in 5ea01805070b
 ---> 7adca3993848
Removing intermediate container 5ea01805070b
Step 3/4 : RUN echo "line2">>test.txt
 ---> Running in a72c23140119
 ---> ed0c8d5187e1
Removing intermediate container a72c23140119
Step 4/4 : CMD cat test.txt
 ---> Running in e5daab213cb3
 ---> 6b0c064170f3
Removing intermediate container e5daab213cb3
Successfully built 6b0c064170f3
Successfully tagged test-isa:latest
```

```
$docker build --tag test-isa .
Sending build context to Docker daemon  2.048kB
Step 1/4 : FROM debian
 ---> be2868bebaba
Step 2/4 : RUN echo "line1">>test.txt
 ---> Using cache
 ---> 7adca3993848
Step 3/4 : RUN echo "line2">>test.txt
 ---> Using cache
 ---> ed0c8d5187e1
Step 4/4 : CMD cat test.txt
 ---> Using cache
 ---> 6b0c064170f3
Successfully built 6b0c064170f3
Successfully tagged test-isa:latest
```

RUN

RUN has 2 forms:

- `RUN <command>` (*shell* form, the command is run in a shell, which by default is `/bin/sh -c` on Linux or `cmd /S /C` on Windows)
- `RUN ["executable", "param1", "param2"]` (*exec* form)

https://docs.docker.com/engine/reference/builder/#run

RUN

RUN has 2 forms:

- `RUN <command>` (*shell* form, the command is run in a shell, which by default is `/bin/sh -c` on Linux or `cmd /S /C` on Windows)
- `RUN ["executable", "param1", "param2"]` (*exec* form)

The `RUN` instruction will execute any commands in a new layer on top of the current image and commit the results. The resulting committed image will be used for the next step in the `Dockerfile`.

https://docs.docker.com/engine/reference/builder/#run

RUN

RUN has 2 forms:

- `RUN <command>` (*shell* form, the command is run in a shell, which by default is `/bin/sh -c` on Linux or `cmd /S /C` on Windows)
- `RUN ["executable", "param1", "param2"]` (*exec* form)

The `RUN` instruction will execute any commands in a new layer on top of the current image and commit the results. The resulting committed image will be used for the next step in the `Dockerfile`.

Layering `RUN` instructions and generating commits conforms to the core concepts of Docker where commits are cheap and containers can be created from any point in an image's history, much like source control.

https://docs.docker.com/engine/reference/builder/#run

RUN + “\”
- wiele komend, jedna warstwa

```
77  RUN set -x \
78          && apt-get update \
79          && apt-get install -y \
80                  ${MONGO_PACKAGE}=$MONGO_VERSION \
81                  ${MONGO_PACKAGE}-server=$MONGO_VERSION \
82                  ${MONGO_PACKAGE}-shell=$MONGO_VERSION \
83                  ${MONGO_PACKAGE}-mongos=$MONGO_VERSION \
84                  ${MONGO_PACKAGE}-tools=$MONGO_VERSION \
85          && rm -rf /var/lib/apt/lists/* \
86          && rm -rf /var/lib/mongodb \
87          && mv /etc/mongod.conf /etc/mongod.conf.orig
88
```

EXPOSE

```
EXPOSE <port> [<port>/<protocol>...]
```

The `EXPOSE` instruction informs Docker that the container listens on the specified network ports at runtime. You can specify whether the port listens on TCP or UDP, and the default is TCP if the protocol is not specified.

EXPOSE

```
EXPOSE <port> [<port>/<protocol>...]
```

The `EXPOSE` instruction informs Docker that the container listens on the specified network ports at runtime. You can specify whether the port listens on TCP or UDP, and the default is TCP if the protocol is not specified.

The `EXPOSE` instruction does not actually publish the port. It functions as a type of documentation between the person who builds the image and the person who runs the container, about which ports are intended to be published. To actually publish the port when running the container, use the `-p` flag on `docker run` to publish and map one or more ports, or the `-P` flag to publish all exposed ports and map them to high-order ports.

VOLUME

```
VOLUME ["/data"]
```

The `VOLUME` instruction creates a mount point with the specified name and marks it as holding externally mounted volumes from native host or other containers. The value can be a JSON array, `VOLUME ["/var/log/"]`, or a plain string with multiple arguments, such as `VOLUME /var/log` or `VOLUME /var/log /var/db`. For more information/examples and mounting instructions via the Docker client, refer to *Share Directories via Volumes* documentation.

VOLUME

```
VOLUME ["/data"]
```

The `VOLUME` instruction creates a mount point with the specified name and marks it as holding externally mounted volumes from native host or other containers. The value can be a JSON array, `VOLUME ["/var/log/"]`, or a plain string with multiple arguments, such as `VOLUME /var/log` or `VOLUME /var/log /var/db`. For more information/examples and mounting instructions via the Docker client, refer to *Share Directories via Volumes* documentation.

The `docker run` command initializes the newly created volume with any data that exists at the specified location within the base image. For example, consider the following Dockerfile snippet:

```
FROM ubuntu
RUN mkdir /myvol
RUN echo "hello world" > /myvol/greeting
VOLUME /myvol
```

This Dockerfile results in an image that causes `docker run` to create a new mount point at `/myvol` and copy the `greeting` file into the newly created volume.

mongo DB - volumeny w Dockerfile

Dockerfile mongoDB - fragment

```
89  RUN mkdir -p /data/db /data/configdb \
90          && chown -R mongodb:mongodb /data/db /data/configdb
91  VOLUME /data/db /data/configdb
92
93  COPY docker-entrypoint.sh /usr/local/bin/
94  RUN ln -s usr/local/bin/docker-entrypoint.sh /entrypoint.sh # backwards compat
95  ENTRYPOINT ["docker-entrypoint.sh"]
96
97  EXPOSE 27017
98  CMD ["mongod"]
```

mongo DB - volumeny

```
docker run --name mongo1 mongo
docker inspect mongo1
```


```
"Mounts": [
        {
            "Type": "volume",
            "Name": "6a1f928c4781510dfa00dd04c1f7c058dd25e7eb813119739822512a41b9a935",
            "Source": "/var/lib/docker/volumes/6a1f928c4781510dfa00dd04c1f7c058dd25e7eb813119739822512a41b9a935/_data",
            "Destination": "/data/db",
            "Driver": "local",
            "Mode": "",
            "RW": true,
            "Propagation": ""
        },
        {
            "Type": "volume",
            "Name": "67269d9bbd5a3201a40e4343d89f0140f156df058d4bde16040da41113694a59",
            "Source": "/var/lib/docker/volumes/67269d9bbd5a3201a40e4343d89f0140f156df058d4bde16040da41113694a59/_data",
            "Destination": "/data/configdb",
            "Driver": "local",
            "Mode": "",
            "RW": true,
            "Propagation": ""
        }
    ],
```

3.1.8. Docker - wady

wady ? - Panie, jakie wady ...

- **Docker containers don't provide bare-metal speed**

  Containers don't have nearly the overhead of virtual machines, but their performance impact is still measurable. If you have a workload that requires bare-metal speed, a container might be able to get you close enough—much closer than a VM—but you're still going to see some overhead.

- A Windows virtual machine can run on a Linux hypervisor and vice versa - Not possible with docker
- **Isolation**

  Since the containers use the same kernel, they are not 100 isolated, so you should be aware of the risks if you are using multiple containers in one server, and make sure you know what you are doing and which containers are running on the same kernel along with your stuff!

wady ? - jakie wady ...

- wciąż relatywnie nowa technologia
- Security - hackers are targeting systems that are hosted in containers and not secured properly.
- problemy z VPN (z doświadczenia własnego)
- brak wsparcia dla aplikacji GUI (choć ludzie kombinują)
- docker-in-docker (choć ludzie kombinują)
- trzeba pamiętać o czyszczeniu śmieci

Docker czyszczenie śmieci

Podczas korzystania z docker'a może się okazać że mamy nieużywane warstwy lub wolumeny które zabierają nam miejsce na dysku a nie są przez nikogo wykorzystywane

docker rmi $(docker images -f "dangling=true" -q)
docker volume rm $(docker volume ls -qf dangling=true)

docker system prune

Zastosowanie

Do czego i jak wykorzystać można dockera?
Do czego i jak wykorzystać można VM?
Jaki problem rozwiązuje docker?

Pytania?

Co jeszcze chcecie wiedzieć?

Wykorzystano materiały

- **docs.docker.com**
- https://blogs.cisco.com/sp/containers-hold-an-agile-approach-to-data-center-virtualization
- https://docs.microsoft.com/en-us/virtualization/windowscontainers/about/
- https://www.infoworld.com/article/3204171/docker/what-is-docker-docker-containers-explained.html
- https://sysadmincasts.com/episodes/31-introduction-to-docker
- https://www.datadoghq.com/docker-adoption/
- What You Need to Know about Docker - Scott Gallagher
- https://osones.com/formations/docker.en.html
- Docker: Up and Running - Karl Matthias & Sean P. Kane

Warto doczytać:

- Dockerfile: polecenie COPY
- docker compose
- Docker-in-docker
- monitorowanie kontenerów dockera
- skalowanie, docker in cloud (Docker Swarm , Kubernetes)
- https://docs.docker.com/develop/develop-images/dockerfile_best-practices/

http://i.imgur.com/ifPgsBH.jpg

Dzieki