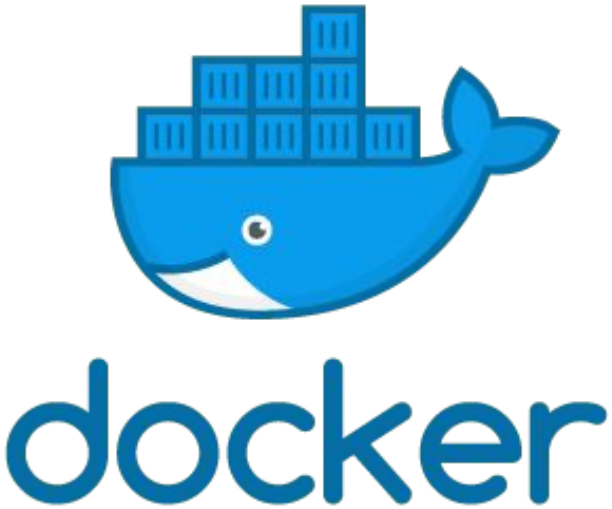
Docker

Dariusz Grabowski



Czym jest docker?

Docker

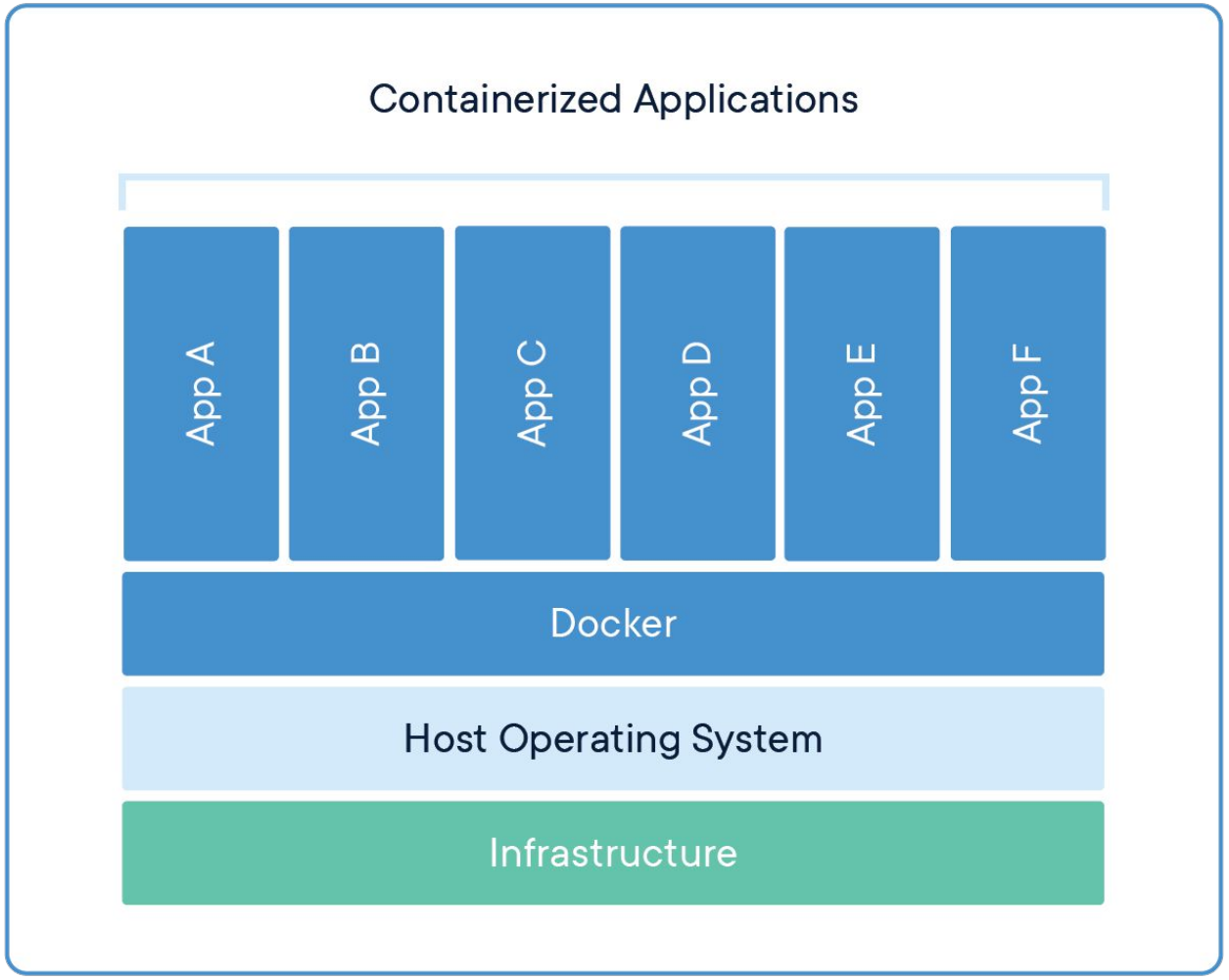
Only independent container platform that enables organizations to seamlessly build, share and run any application, anywhere—from hybrid cloud to the edge.

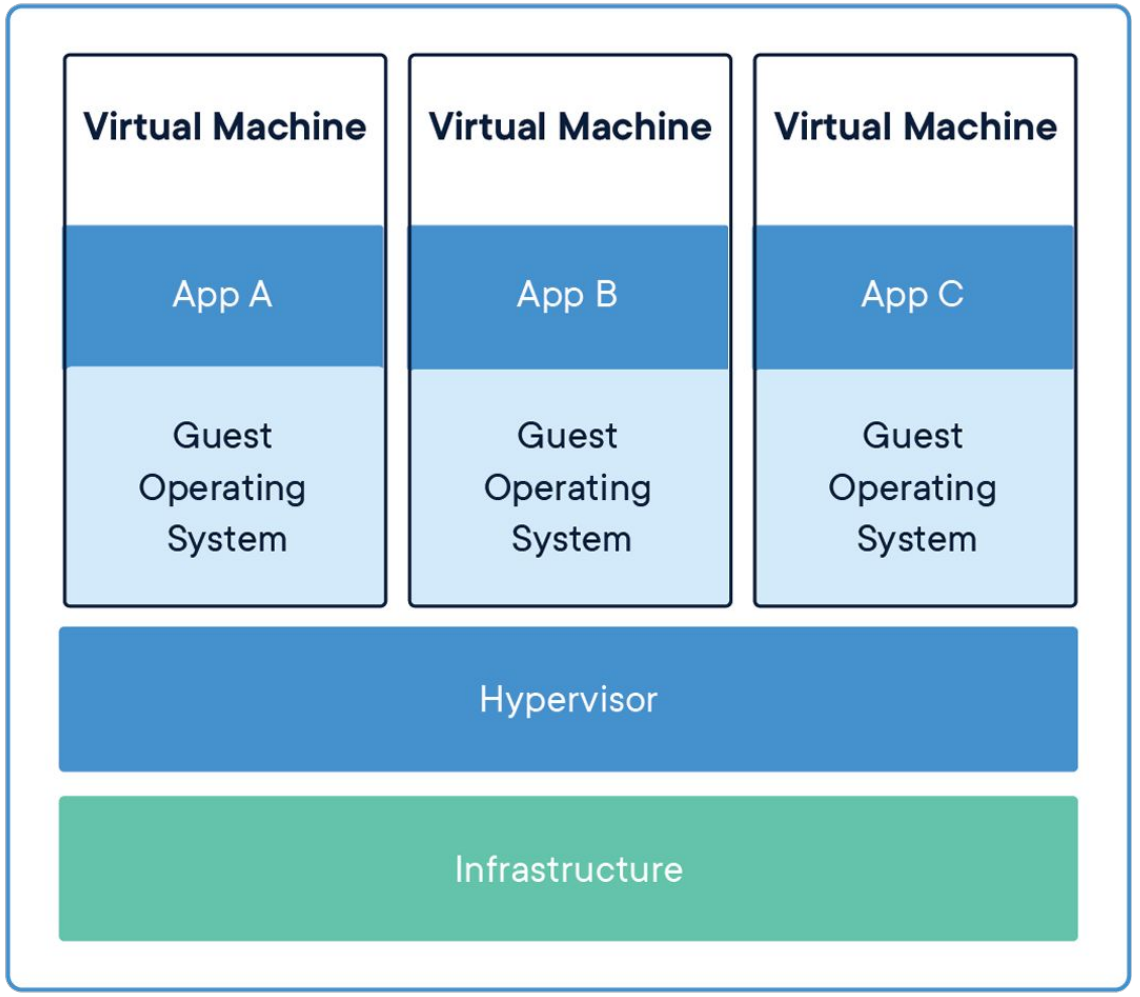
- docker.com

Otwarte oprogramowanie służące do realizacji wirtualizacji na poziomie systemu operacyjnego (tzw. "konteneryzacji"), działające jako „platforma dla programistów i administratorów do tworzenia, wdrażania i uruchamiania aplikacji rozproszonych”.

- wikipedia.pl

Docker vs VM

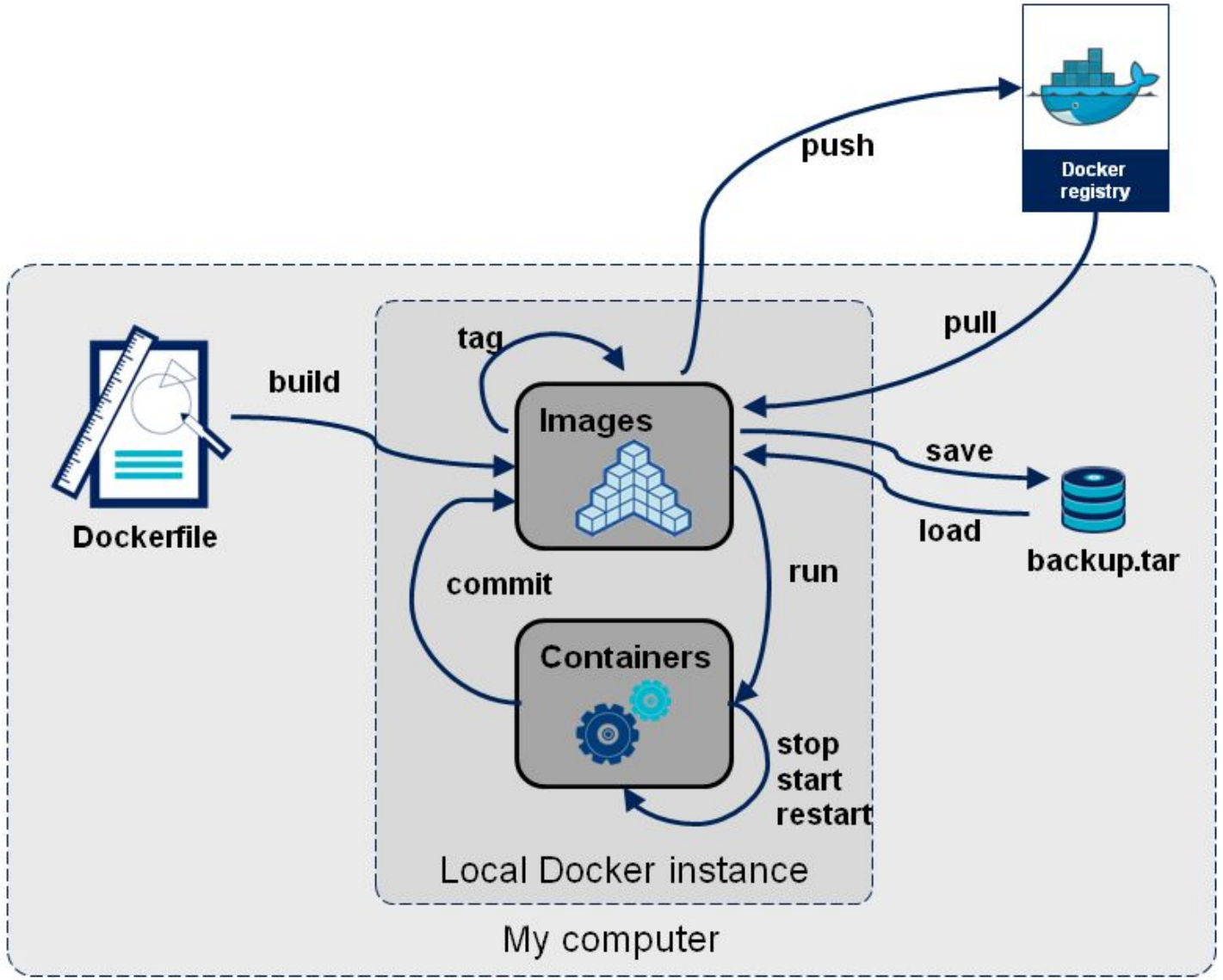

Containerized Applications
App A
App B
App C
App D
App E
App F
Docker
Host Operating System
Infrastructure


Virtual Machine
Virtual Machine
Virtual Machine
App A
App B
App C
Guest Operating System
Guest Operating System
Guest Operating System
Hypervisor
Infrastructure

Docker

- Flexible: Nawet najbardziej skomplikowane aplikacje mogą być skonteneryzowane
- Lightweight: Kontenery współdzielą kernel hosta
- Interchangeable: Możliwe jest wdrażanie aktualizacji w locie
- Portable: zbudowany lokalnie kontener może zostać wdrożony na dowolnym innym hoście
- Scalable: Możliwe jest zwiększenie ilości kontenerów i rozdystrybuowanie ruchu pomiędzy nimi
- Stackable: Możliwe jest dodawanie funkcjonalności do już wcześniej zbudowanych kontenerów

Docker



Za https://blog.octo.com/en/docker-registry-first-steps/

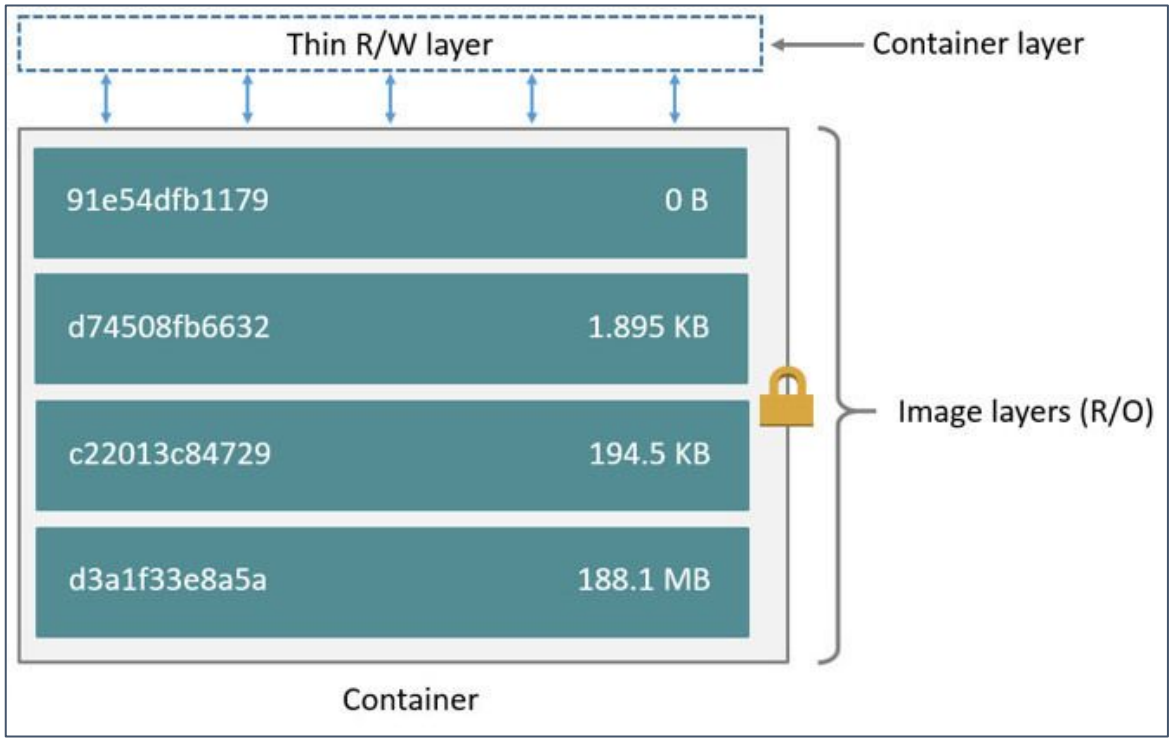
Docker polecenia

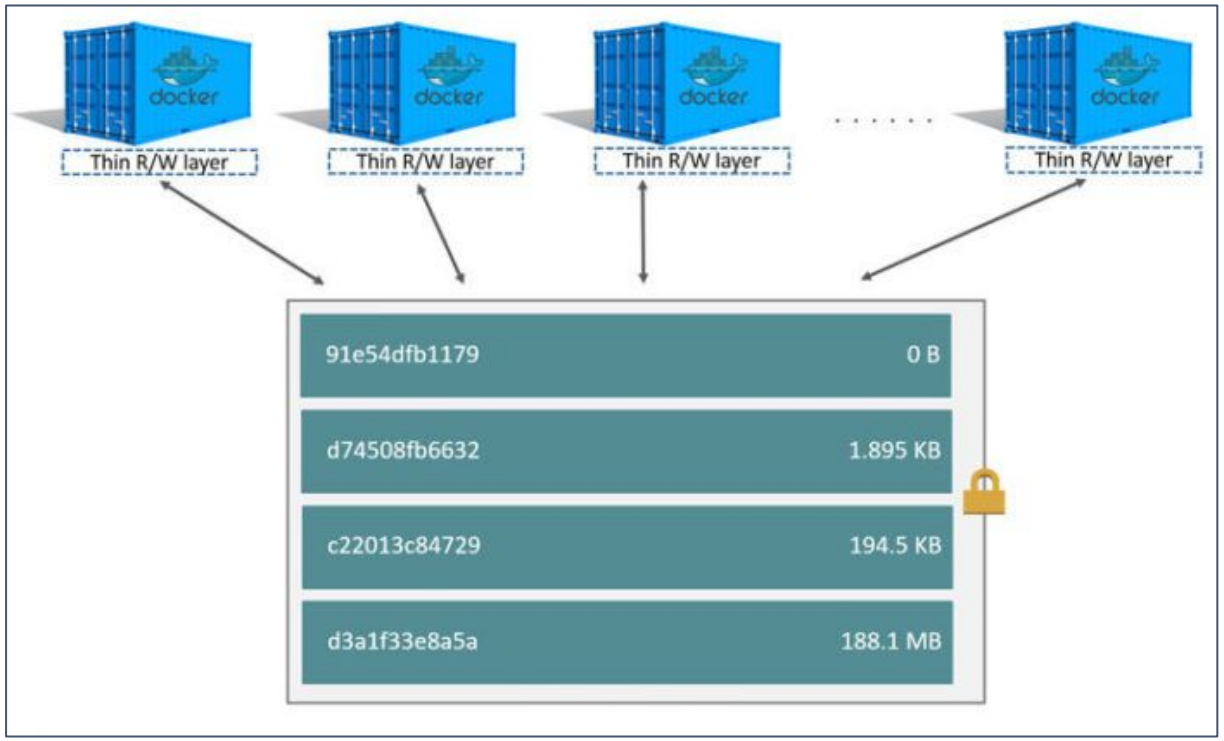
```
# uruchomienie i zamknięcie kontenera
docker run hello-world

# uruchomienie kontenera o nazwie “ubuntu” w trybie interaktywnym,
usunięcie po zakończeniu pracy
docker run -it --rm ubuntu /bin/bash

# wyświetla listę obrazów w lokalnym cache
docker images

# wyświetla listę kontenerów łącznie z zatrzymanymi
docker container ls -a

# zbudowanie obrazu na bazie pliku Dockerfile z bieżącego katalogu
docker build . -t myimage:latest
```

Dockerfile

```
# Używamy oficjalnego obrazu ubuntu jako obrazu bazowego
FROM ubuntu:19.10

# Ustawiamy katalog roboczy w obrazie /app
WORKDIR /app

# instalujemy niezbędne pakiety
RUN apt-get update && apt-get install -y g++ gcc

# Kopiujemy zawartość bieżącego kat. na hoście do /app
COPY . /app

# Ustawiamy zmienną środowiskową
ENV MY_VARIABLE hello

# kompilacja
RUN g++ main.cpp

# Uruchamiamy binarkę
CMD ["./a.out"]
```

Docker warstwy





Za https://www.shellhacks.com/docker-image-vs-container/

Dockerfile

Parametry docker run:

-d - uruchomienie jako deamon

--name <nazwa> - ustawianie nazwy dla kontenera

--rm - usunięcie po zatrzymaniu

-it - tryb interaktywny

--ip="" - ustawienie adresu IP kontenera

--cpus=0.000 - ilość procesorów dla kontenera

--memory="" - limit pamięci. Minimum 4M.

-v /host/directory:/app - montowanie katalogu z hosta w kontenerze

-p 4000:80 - wystawianie portu kontenera na zewnątrz

Dockerfile dobre praktyki

- Używaj oficjalnych obrazów kiedy to tylko możliwe
- Używaj jak najmniej poleceń - każde z nich tworzy nową warstwę.
- Łącz polecenia poprzez && np.

  ```
  apt-get update && apt-get install ssh
  ```

- Instaluj tylko te pakiety które są niezbędnie potrzebne
- Kopiowanie plików umieść na końcu Dockerfile
- Używaj jak najmniejszych obrazów (np. dystrybucja alpine)

Pytania?

Zadanie

1. Zbudować obraz do cross kompilacji dla wybranej architektury (mips, arm, aarch64, powerpc)

   W Dockerfile należy:

   a. wybrać bazowy obraz np ubuntu, alpine, centos

   b. zainstalować kompilator np: gcc-arm-linux-gnueabihf g++-arm-linux-gnueabihf

   c. zainstalować conana

   d. zainstalować cmake'a (v. 3.15)

2. Stworzyć plik cmake toolchain lub dodać zmienne środowiskowe w conan profile

   (polecenie `conan profile update...`)

3. Zbudować paczkę conanową w uruchomionym kontenerze

4. `git clone https://github.com/grabowski-d/docker-cross-compiler.git`

Dodatkowo: podczas budowania paczki powinny uruchomić się testy paczki (z test_package).

Dlaczego to powinno się udać pomimo innej architektury?