Konteneryzacja aplikacji – Docker

Marcin Szeląg, IIn, PP

Referencje

- https://www.docker.com/
- https://docs.docker.com/engine/docker-overview/
- https://docs.docker.com/get-started/ - dokumentacja Dockera
- https://www.youtube.com/watch?v=Q5POuMHxW-0 (Introduction to Docker; Solomon Hykes)
- https://www.youtube.com/watch?v=pGYAg7TMmp0 (Docker Tutorial - What is Docker & Docker Containers, Images, etc?; LearnCode.academy)

Referencje

- https://www.youtube.com/watch?v=JBtWxj9I7zM (Docker Tutorial - Docker Container Tutorial for Beginners; LearnCode.academy)
- https://www.youtube.com/watch?v=K6WER0oI-qs (Docker Container Tutorial - How to build a Docker Container & Image; LearnCode.academy)
- https://youtu.be/wCTTHhehJbU (Learn Docker in 20 Minutes; Dylan Israel)
- https://www.youtube.com/watch?v=JprTjTViaEA (From Zero to Docker - Tutorial for Beginners; Jonny L)

docker

Wprowadzenie

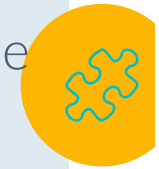
https://upload.wikimedia.org/wikipedia/commons/7/79/Docker_%28container_engine%29_logo.png

Wprowadzenie

- pierwsze wydanie w **2013** r.
- wersja darmowa – **Docker Desktop Community Edition** (CE)
- Działa na różnych systemach operacyjnych: Linux, Mac OS, Windows
- dla **Windows 10**: https://store.docker.com/editions/community/docker-ce-desktop-windows (korzysta z natywnego hipernadzorcy Hyper-V do stworzenia maszyny wirtualnej MobyLinuxVM)
- dla **Mac OS**: https://hub.docker.com/editions/community/docker-ce-desktop-mac
- wersje dla Linuxa, AWS, Azure, …

Wprowadzenie - założenia

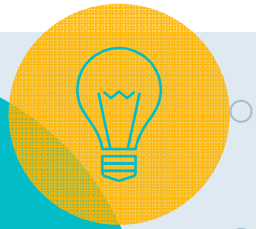
- Standaryzacja **wdrażania aplikacji** (***software deployment***) w środowisku produkcyjnym
- „**Konteneryzacja**” aplikacji - eliminacja sytuacji „u mnie działa”
- **Kontener** – zawiera **aplikację** + **środowisko** potrzebne do jej działania (biblioteki, zmienne środowiskowe, OS, skrypty, …)
- Analogia do świata rzeczywistego – problem wysyłki towarów do klientów => wprowadzenie wysyłki w **kontenerach** (podział odpowiedzialności pomiędzy producenta i dostawcę)
- Kontener jako „***unit of software delivery***”



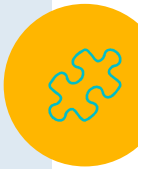
Wprowadzenie - założenia

https://medium.freecodecamp.org/a-beginner-friendly-introduction-to-containers-vms-and-docker-79a9e3e119b

Wprowadzenie - założenia



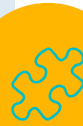
- Umożliwienie realizacji koncepcji architektury aplikacji opartej o **mikrousługi** (*microservices*)
- Docker definiuje format tzw. **kontenera** (*container*) zawierającego **aplikację** oraz **jej wszystkie zależności**.
- Kontener może być uruchamiany na **dowolnej maszynie z systemem Docker**
- Docker gwarantuje **identyczność środowiska wykonawczego aplikacji** na każdej z tych maszyn, w szczególności w trakcie rozwoju aplikacji (*development*), jej testowania i po jej wdrożeniu (udostępnieniu użytkownikom końcowym)



Wprowadzenie - założenia

Zalety konteneryzacji aplikacji:

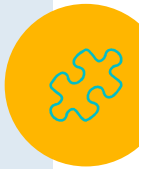
- **Elastyczność**: nawet najbardziej skomplikowane aplikacje mogą działać w architekturze komunikujących się kontenerów.
- **Lekkość**: kontenery wykorzystują i współdzielą jądro systemu operacyjnego hosta.
- **Wymienność**: możliwość wdrażania aktualizacji działającej aplikacji „w locie”.



Wprowadzenie - założenia

Zalety konteneryzacji aplikacji:

- **Przenośność**: możliwość uruchamiania aplikacji lokalnie i w chmurze.
- **Skalowalność**: możliwość automatycznego zwiększania liczby replik kontenerów i ich dystrybuowania między węzłami obliczeniowymi.
- **Interoperacyjność**: możliwość uruchamiania usług składowych aplikacji na różnych maszynach.

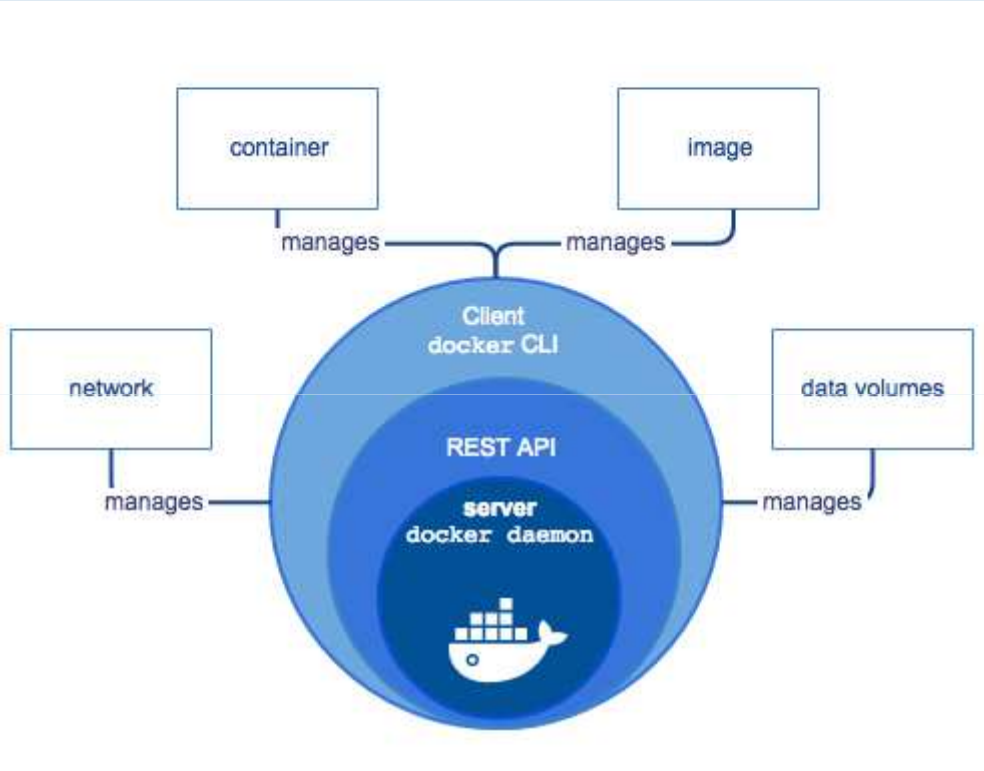


docker

Architektura

https://upload.wikimedia.org/wikipedia/commons/7/79/Docker_%28container_engine%29_logo.png

Docker Engine



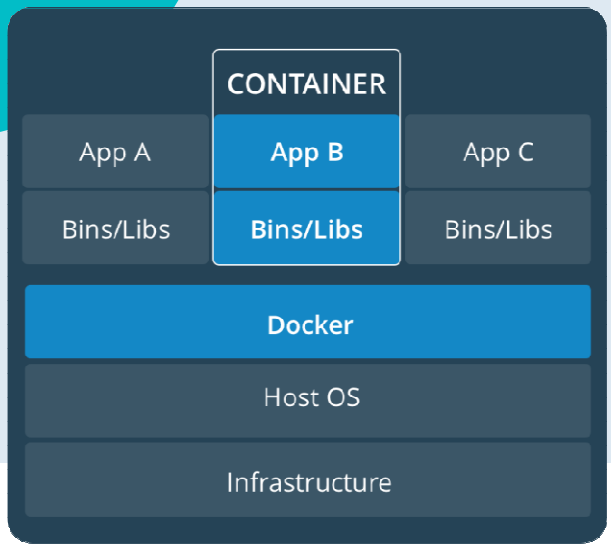
https://docs.docker.com/engine/images/engine-components-flow.png

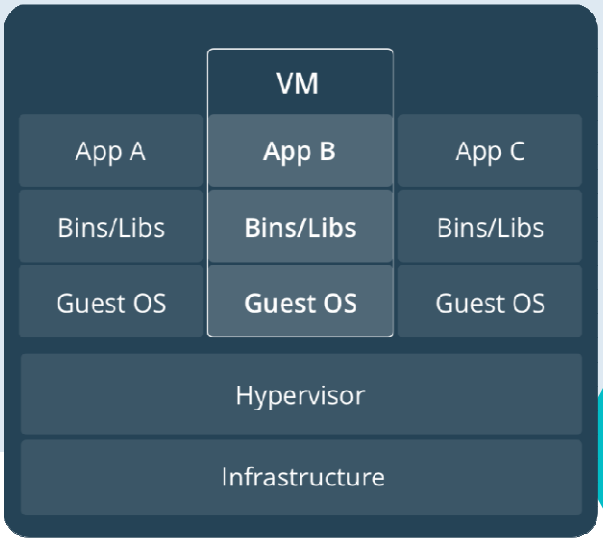
Obrazy i kontenery

- **Obraz** (*image*) – uruchamialny pakiet zawierający elementy konieczne do uruchomienia aplikacji: kod aplikacji, środowisko uruchomieniowe aplikacji, biblioteki, ustawienia zmiennych środowiskowych, pliki konfiguracyjne
- **Kontener** (*container*) – instancja obrazu w pamięci operacyjnej (proces)
- `docker run -i -t ubuntu /bin/bash`
- Możliwość uruchomienia **wielu kontenerów** na tym samym komputerze (z tego samego lub z różnych obrazów)
- Możliwość użycia jednego obrazu jako **bazy** do stworzenia innego obrazu (**FROM <parent-image>**)
- Zbudowane obrazy –> **rejestr** (*docker registry*)

Kontenery

- **Kontener vs. maszyna wirtualna** (https://docs.docker.com/get-started/#containers-and-virtual-machines) – maszyny wirtualne często zużywają za dużo zasobów, w stosunku do zapotrzebowania aplikacji, są zbyt „ciężkie”



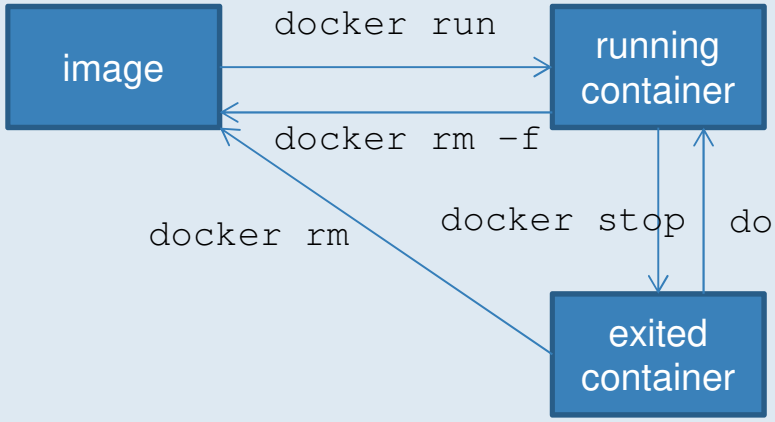


Data volumes, networks

- Kontenery ułatwiają komunikację na linii **programiści** (developers) – **operatorzy** (ops) zajmujący się eksploatacją oprogramowania (operations)
- **Data Volume** – **mapowanie** pomiędzy katalogiem systemowym a katalogiem widzianym przez uruchomiony obraz (kontener) Dockera; zapewnienie **trwałości danych** (persistence), np. zapisywanych w kontenerze do bazy danych
- **Sieć** (bridge network) – umożliwia komunikację między kontenerami (https://docs.docker.com/network/bridge, https://docs.docker.com/config/containers/container-networking)

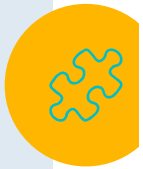
Stany kontenera

- **Stany kontenera**:
  - **nieuruchomiony** (obraz)
  - **uruchomiony** (*up*) (wyświetlanie: `docker ps`)
  - **zatrzymany** (*exited*) – pamięta dane (wyświetlanie: `docker ps --all`)
  - **usunięty**

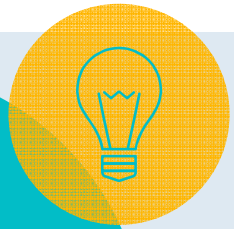


Docker Deamon, Docker Client

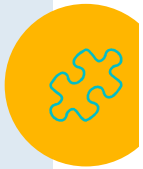
- **Docker Deamon** – usługa działająca w tle na komputerze-hoście, zarządzająca budowaniem, uruchamianiem i dystrybuowaniem kontenerów Dockera
- **Docker Client** – narzędzie dostępne z wiersza poleceń, np. `Windows PowerShell` czy `cmd`, umożliwiające użytkownikowi interakcję z Docker Deamon'em



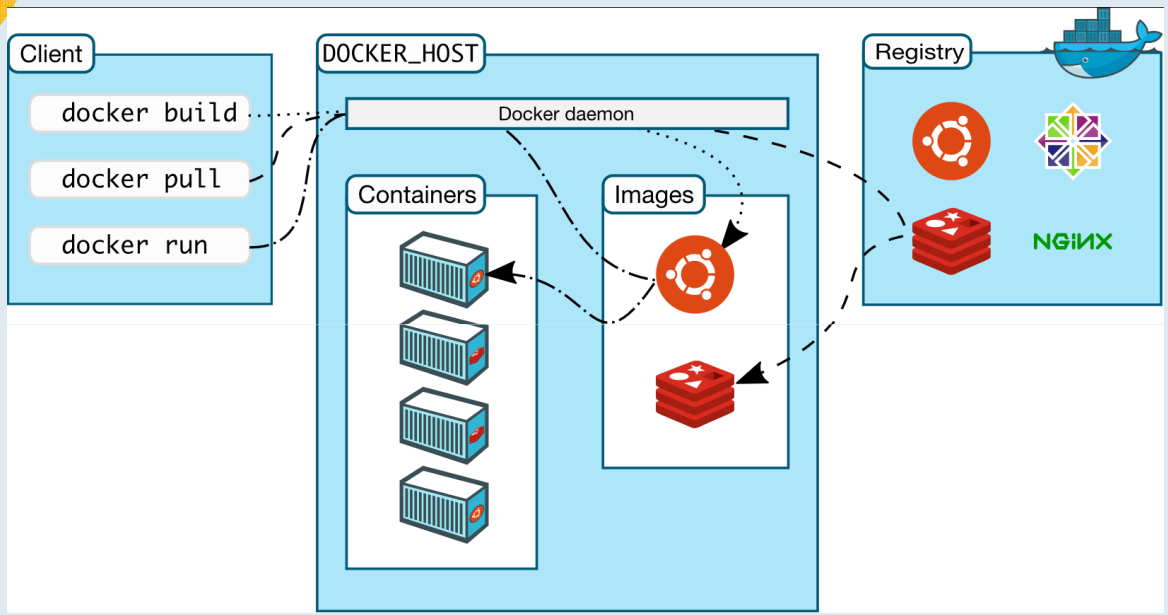
Rejestr obrazów (Docker registry)



- **Docker Hub**: https://hub.docker.com/ - domyślny centralny rejestr (publicznych) obrazów Docker'a, umożliwiający ich współdzielenie (trzeba stworzyć **Docker ID**)
- Możliwość uruchomienia **własnego rejestru** (prywatnych) obrazów Docker'a
- (https://docs.docker.com/registry)
- `docker pull`
- `docker push`
- **Docker Store** – część obrazów płatna
- **Rejestr** = kolekcja repozytoriów
- **Repozytorium** = kolekcja obrazów (różne tagi)
- **`docker login/logout – logowanie do rejestru`**



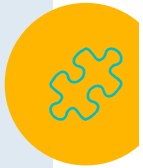
Komunikacja



https://docs.docker.com/engine/images/architecture.svg

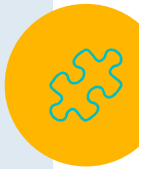
Obraz (*image*)

- Wzorzec kontenera (read-only); „**przepis**”
- Zwykle **bazuje na innym obrazie** i definiuje dodatkowe akcje
- Gotowe obrazy – **rejestr**
- Własne obrazy – **`Dockerfile`** (kroki potrzebne do stworzenia obrazu)
- Każda linia (krok) w **`Dockerfile`** odpowiada **warstwie** (*layer*), zarządzanej przez Docker’a



Obraz (*image*)

- **`docker build -t my-application .`** – stworzenie obrazu na podstawie pliku `Dockerfile` w aktualnym katalogu
- **inkrementalne budowanie** (tylko zmienione warstwy)
- **`docker images`**, **`docker image ls`** – lista obrazów w lokalnym rejestrze Dockera
- **`docker push`** – wgranie obrazu z lokalnego do zdalnego rejestru
- **`docker rmi`, `docker image rm`** – usuwanie obrazów
- **`docker tag`** – nadanie taga (aliasu) istniejącemu obrazowi

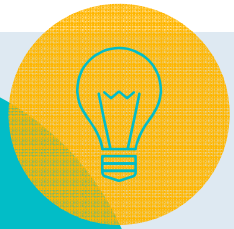


Obraz (*image*)

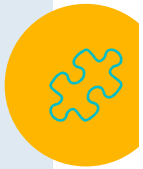
```
> docker tag openjdk my-java
> docker tag registry.com/i/rank-generator
  registry.com/i/rg:1.6.0
> docker images

REPOSITORY                          TAG     IMAGE ID
openjdk                             latest  831a029b6add
my-java                             latest  831a029b6add
registry.com/i/rank-generator       latest  3028492acb76
registry.com/i/rg                   1.6.0   3028492acb76
```

Dockerfile – przykład 1

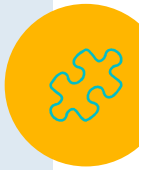



```
FROM openjdk:8-jre-alpine
LABEL maintainer=marcin.szelag@cs.put.poznan.pl
COPY rule-inducer-1.5.0.war /app/rule-
    inducer.war
COPY start.sh /app/start.sh
RUN apk --update --no-cache add curl
RUN apk --update --no-cache add bash
ENTRYPOINT ["/bin/sh"]
CMD ["/app/start.sh"]
EXPOSE 8009
HEALTHCHECK CMD curl -f
    http://localhost:8009/health || exit 1
```



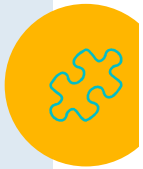
Dockerfile – przykład 2

```
FROM python:2.7-slim
WORKDIR /app
ADD . /app
RUN pip install --trusted-host pypi.python.org -r
    requirements.txt
EXPOSE 80
ENV NAME World
# Run app.py when the container launches
CMD ["python", "app.py"]
```

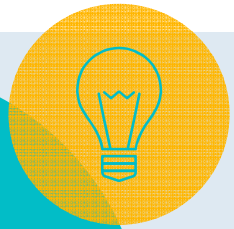


Kontener

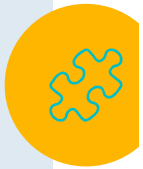
- Instancja obrazu
- Konfigurowany przy uruchomieniu za pomocą **parametrów uruchomieniowych**
- Może być **podłączony do 1 lub więcej sieci**
- Może mieć podłączony **Data Volume**
- Aktualny stan może zostać zapisany jako **nowy obraz**
- Domyślnie **izolowany** od hosta i innych kontenerów
- Po usunięciu wszelkie zmiany stanu (niezapisane w sposób trwały) są **tracone**
- **`$ docker run -i -t ubuntu /bin/bash`**



Kontener

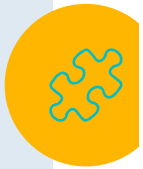


- możliwość śledzenia kolejnych wersji kontenerów, w stylu Git-a (docker commit, docker diff)
- **`docker ps (--all)`**, **`docker container ls (--all)`**
- tryb pracy interaktywny (-i -t)
- tryb pracy w tle (-d), możliwość podłączenia się do kontenera (**`docker attach`**), możliwość odłączenia się od kontenera bez jego zatrzymywania



Kontenery Linux a kontenery Windows

- **Kontenery dla Linuxa** (Linux containers) – można je uruchomić na Docker Engine działającym w systemie Linux (może to być również host wirtualny, np. Moby Linux VM zarządzany przez Hyper-V w Win 10 Pro)
- **Kontenery dla Windowsa** (Windows containers) - można je uruchomić na Docker Engine działającym w systemie Windows
- Możliwość przełączania w Docker Desktop for Windows; eksperymentalnie: LCOW (Linux Containers on Windows)



docker

Przykładowe Komendy

https://upload.wikimedia.org/wikipedia/commons/7/79/Docker_%28container_engine%29_logo.png

Dziękuję!

Pytania?