docker

docker

Docker - dlaczego?

- ułatwia / przyspiesza cykl wytwarzania
- możliwość szybkiego uruchamiania programu w różnych środowiskach
- ułatwia develop. / test / deploy w rozproszonych app.
- w skrócie: zwiększa wydajność

Docker - edycje

- **linux** vs **Mac/Win** vs **chmura**(AWS/Azure/Google)
- **CE** (darmowa) vs **EE** (płatna, support, extras)
- **Edge**(beta, miesiąc) vs **Stable**(kwartał)

Kontener vs VM

- **kontenery to** nie mini-VM
- to **procesy**
- o ograniczonym dostępie do zasobów (pliki, sieć, urządzenia, inne procesy)

Utworzenie pierwszego kontenera

- instalacja, sprawdzenie wersji docker’a
- utworzenie **kontenera** **z** **obrazu**
- poznanie podstawowych komendy docker’a:
(run, start, stop, ps, logs, rm)

Obraz vs kontener

- **obraz to aplikacja**, którą chcemy uruchomić
- **kontener to** uruchomiony **proces** - instancja tego obrazu
- możesz mieć wiele kontenerów uruchomionych z tego samego obrazu
- główne repo obrazów dla docker’a: **Docker Hub**

docker container run -p 80:80 nginx

docker

Co się wydarzyło?

Docker:

- **wyszukał obraz** w lokalnym buforze
- nie znalazł, **załadował** z Docker Hub najnowszy
- **utworzył nowy kontener** na podstawie obrazu
- **przypisał** wirtualne **IP** w prywatnej, wewnętrznej sieci
- **otworzył port** 80 **na hoście** i **przekierował** na 80 kontenera
- **uruchomił kontener** korzystając z CMD w Dockerfile obrazu

docker run vs start

- run tworzy nowy kontener
- start uruchamia istniejący
- stop zatrzymuje
- jeśli używamy **id** kontenera z komendą (run, start etc.) zazwyczaj wystarczą pierwsze 3 znaki

1 kontener - 1 port na hoście

docker container run --name my_second_container -p 8888:80 nginx

Ćwiczenie I

TODO

- zaprzyjaźnij się z **--help** i **docs.docker.com :)**
- **wyświetl listę kontenerów** (ps / ls [-a])
- do wystartowania **3 kontenery**
- **nginx**, **httpd**, **mysql**
- **każdy** z nich **w tle** (-d)
- **każdy** z nich **z nadpisaną nazwą** (--name)
- porty dla serwerów:
  - **nginx 80:80**
  - **httpd 8080:80**
  - **mysql 3306:3306**
- dla mysql'a przekaż parametr MYSQL_RANDOM_ROOT_PASSWORD=yes (--env)
- **wyświetl listę kontenerów**
- sprawdź w logach mysql GENERATED ROOT PASSWORD
- **usuń utworzone kontenery**
- **wyświetl listę kontenerów**

Utworzenie kontenerów

docker container ls (-a)

docker container run -d -p 80:80 --name the_nginx nginx

**docker container run -d -p 8080:80 \
--name the_httpd httpd**

**docker container run --name the_mysql -d -p 3306:3306 \
--env MYSQL_RANDOM_ROOT_PASSWORD=yes mysql**

docker container ls (-a)

Logi w mysql

**docker container logs ID_KONTENERA 2>/dev/null **
l grep "GENERATED ROOT PASSWORD”

lub docker logs ID_KONTENERA i wyszukać w wyświetlonym logu

Usunięcie kontenerów

docker stop the_nginx the_httpd the_mysql
 # jeśli bez stop, przy rm flaga -f

docker rm the_nginx the_httpd the_mysql

 # poniżej niebezpieczna alternatywa - wszystkie!
docker rm -f $(docker ps -aq)

docker container ls (-a)

Podglądanie kontenera

- **docker container top** - procesy w kontenerze
  na Win / Mac zachowuje się inaczej niż na linux

- **docker container inspect** - szczegóły konfiguracji

- **docker container stats** - CPU, MEM etc.

Konsola w kontenerze

- **docker container run -it ubuntu bash**
- **docker container exec -it <id> bash**
- **docker container attach <id>**
- **# exit vs Ctrl + PQ**

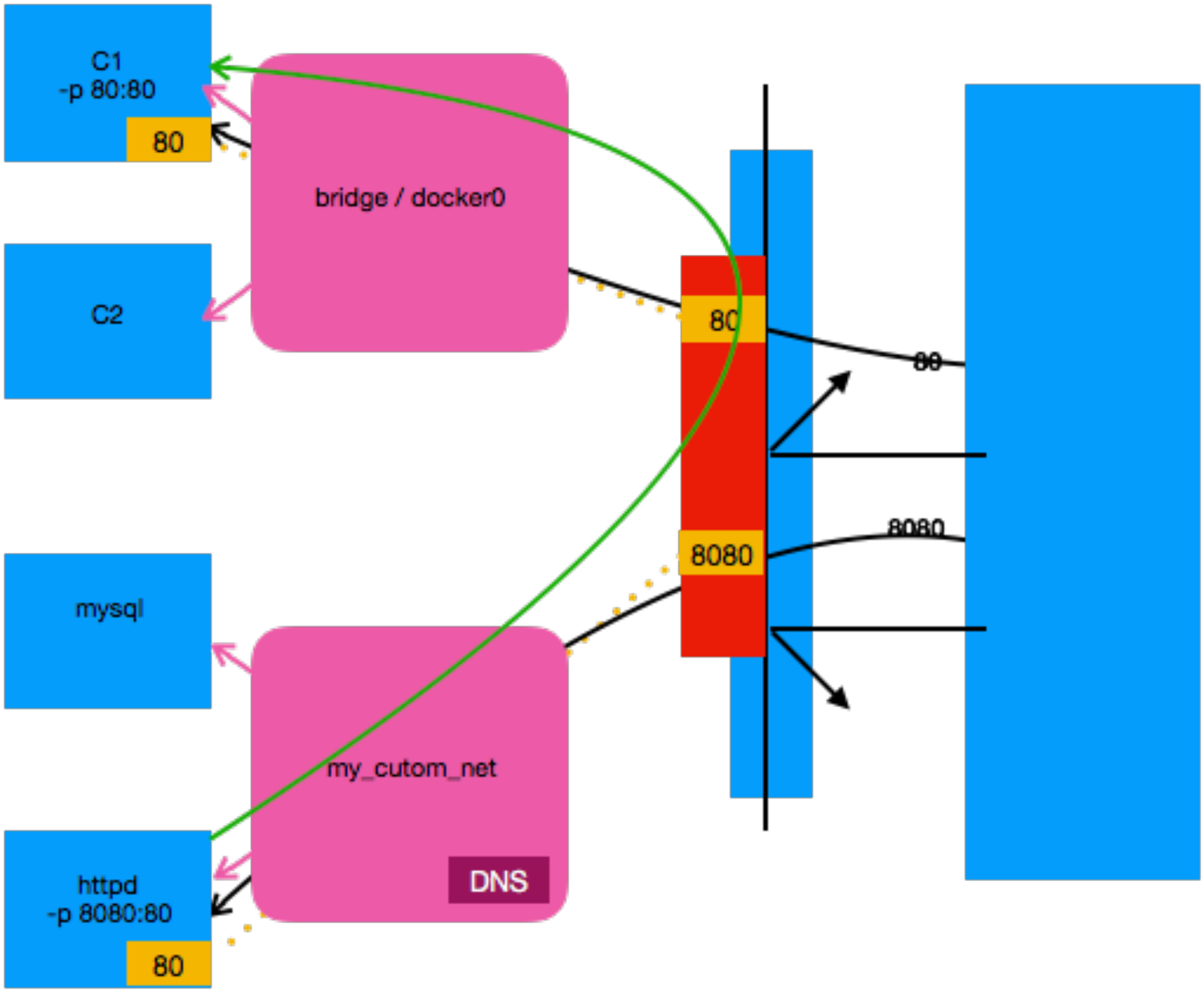
Podanie komendy nadpisuje default CMD.
Np. w przypadku run ubuntu, bash to default - nie potrzeba dodawać.

Docker i sieć - default

- każdy **kontener podłączany do wirtualnej sieci**
- wszystkie **kontenery w ramach prywatnej sieci** mogą komunikować się ze sobą bezpośrednio (**bez wystawiania portów**)
- w zależności od potrzeb, **można zmienić** zachowanie ze sterownika **bridge** (**domyślny**) na host, overlay, macvlan, none, lub 3rd-party plugin

docker

Docker i sieć



Docker i sieć - DNS

- Kontenery **nie powinny używać** dla komunikacji **IP**
- docker ma **wbudowany server DNS**
- utworzona, **własna sieć**: **automatic DNS resolution**, dla kontenerów w tej sieci
- w domyślnej sieci (bridge) nie ma DNS'a (można użyć --link, ale lepsza własna sieć)
- **domyślna nazwa** - **nazwa kontenera** (można użyć alias)

Docker i sieć - komendy

- **docker network ls**
- **docker network inspect --format ‘{{.NetworkSettings}}'**
- **docker network create --driver**
- **docker network connect**
- **docker network disconnect**
- **docker container port**

Ćwiczenie II - konsola w kontenerze

TODO:

- 2 kontenery z dystrybucjami linux: ubuntu i centos
- instalacja curl w kontenerze i sprawdzenie jego wersji
- ubuntu: apt-get update && apt-get install curl
- centos: yum update curl
- przy uruchomieniu użyj opcji --rm (oszczędzenie czasu na sprzątanie)

Konsola w kontenerze - rozwiązanie

- docker container run -it --rm ubuntu
- #apt-get update && apt-get install curl
- #curl —version
- #exit
- docker container run -it --rm centos
- # yum update curl
- #curl —version
- #exiit

Ćwiczenie III - DNS

- utwórz własną sieć
- wystartuj 2 kontenery nginx:alpine (detached)
- przetestuj (użyj exec) ping z jednego kontenera do drugiego, używając domyślnego DNS’a dla danego kontenera

DNS - rozwiązanie

- docker network create my_network
- docker container run --name alpine1 -d --network=my_network nginx:alpine
- docker container run --name alpine2 -d --network=my_network nginx:alpine
- docker network inspect my_network
- docker container exec -it alpine1 ping alpine2

DNS Round Robin

- DNS Round Robin - wiele adresów IP, ukrytych za jedną nazwą DNS
- od wersji 1.11 DNS RR w dockerze ‘za darmo’

Ćwiczenie IV - DNS Round Robin Test

TODO :

- utworzyć sieć prywatną (domyślny sterownik)
- utworzyć w powyższej sieci dwa kontenery
    - z aliasem DNS ‘search’
    - serwerem elasticsearch
- test curl’em :
    - serwer na domyślnym porcie 9200 zwraca JSON’a z wygenerowaną nazwą serwera
    - powinniśmy przy kilku request’ach, używających tej samej nazwy, uzyskać 2 różne nazwy serwera
- użyj obrazu elasticsearch:2
- do testów można użyć dystrybucję centos (ma curl’a)
    - ten kontener też trzeba wpiąć w sieć

DNS Round Robin Test - rozwiązanie

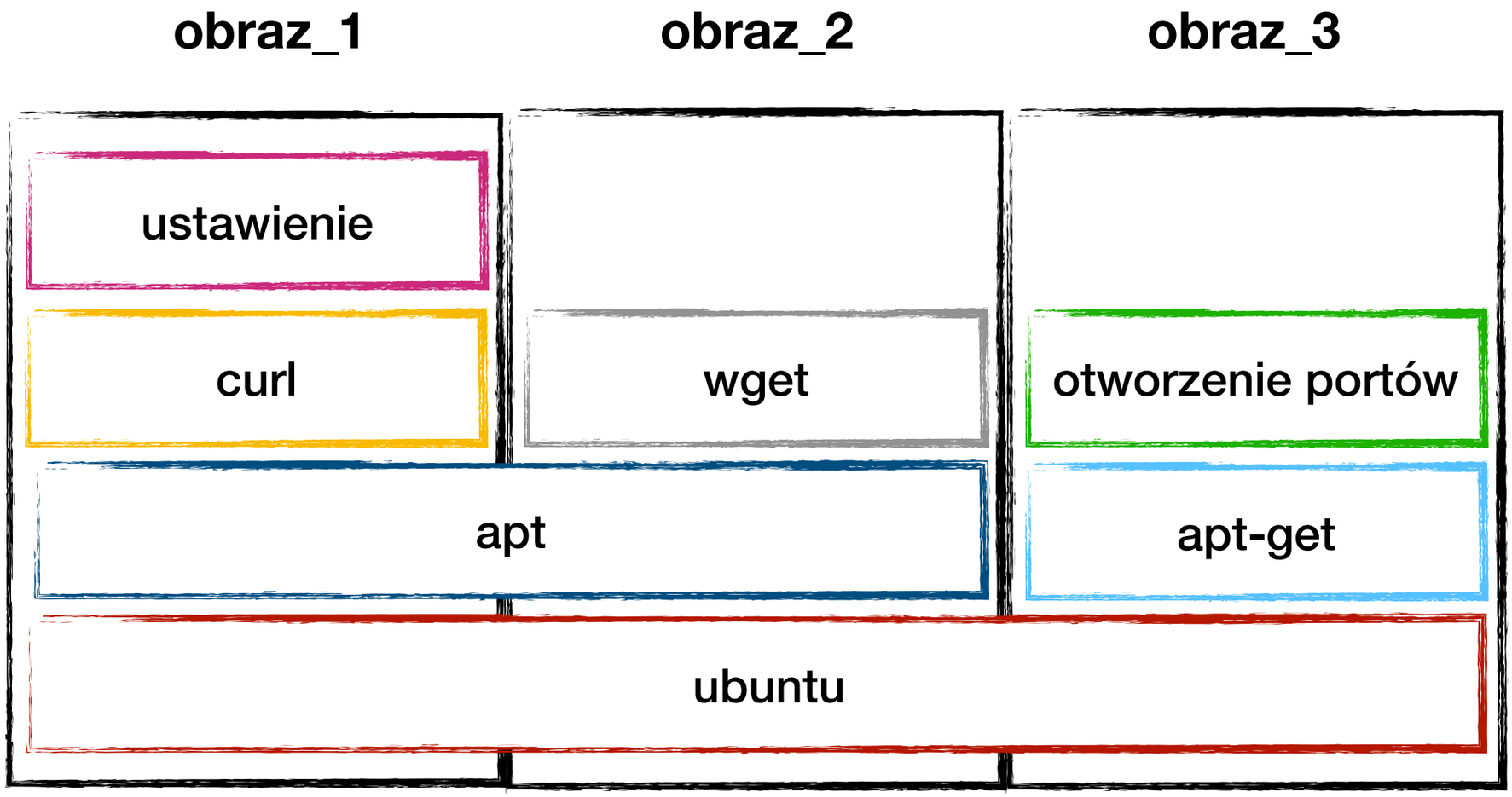
- docker network create robin_net
- docker container run -d --name es1 [es2] --net robin_net --net-alias search elasticsearch:2
- docker container run --rm --net robin_net centos curl -s search:9200

czasem trzeba kilka razy curlem ‘strzelić’ żeby zmienić serwer

obraz

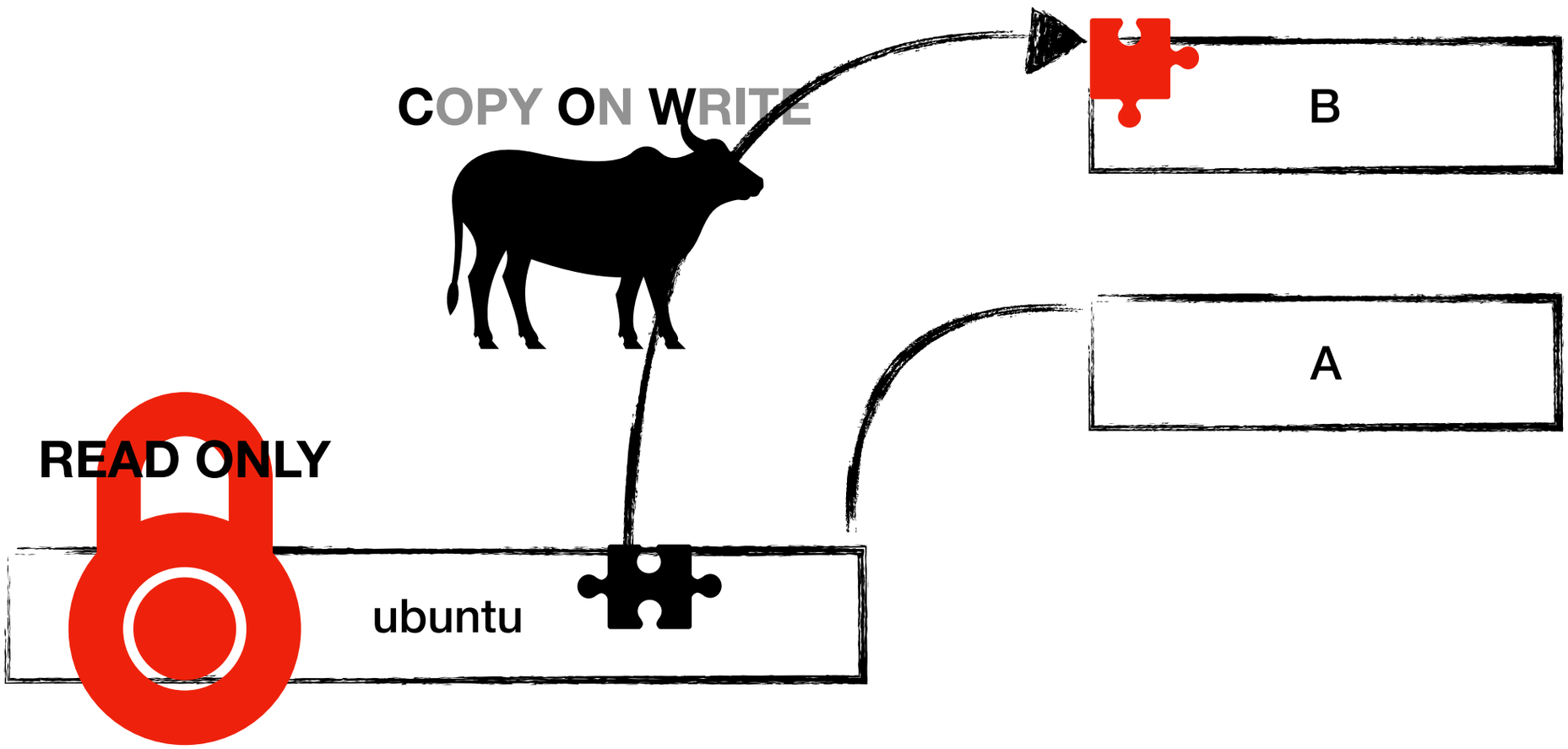
- obraz - czym jest
  - ‘binarki’ aplikacji i potrzebne zależności
  - metadane jak je ruchamiać
- obraz - czym nie jest
  - kompletnym OS’em

warstwy



- sha256: ubuntu, apt, curl etc. zapisywane na hoście (Docker) tylko raz

COW



- A - system plików identyczny jak na ubuntu
- B - system plików ze zmianą

docker

image - komendy

- docker image ls [-a]
- docker image history <obraz>
- docker image inspect <obraz>
- docker image build [-f custorm-Dockerfile-name]

Dockerfile 

- FROM - zazwyczaj z parent image
  - można od 0 (scratch)
- ENV - zmienne systemowa
- RUN - komendy uruchamiane podczas budowania kontenera
- EXPOSE - wystawienie portów
  - w wirtualnej sieci
  - otwarcie na hoście
    - flaga -p podczas docker run lub
    - ports w docker-compose
- CMD - zdefiniowanie, co będzie odpalone przy uruchomieniu kontenera

volumes & bind mouts

- volumes
  - utworzenie zasobu poza UFS kontenera
    - podpięcie do kontenera

-v [name:] /container/path

-v path:/container/path

- bind mouts -v /host_path:/container_path
  - link ścieżki w hoście ze ścieżką w kontenerze
- w obu przypadkach kontener ‘widzi’ lokalny zasób

volumes

- sposoby konfiguracji
  - **Dockerfile** : VOLUME
    - VOLUME /var/lib/mysql
      - utworzy na hoście zasób
        - po usunięciu kontenera zasób nadal będzie istniał
          - aż do usunięcia
- docker **container run** -v /var/lib/mysql
  - container run -v my_volume:/var/lib/mysql
- docker **volume create**
  - gdy chcemy ‘tuningować’ driver

docker container inspect

- “Mounts”
  - “Destination”: co widzi kontener
  - “Source”: zasób na hoście
    - na linuxie znajdziemy ten zasób
    - na Win / macOS - nie
      - linuxowa VM

bind mounting

- mapowanie zasobu hosta do zasobu kontenera
- w skrócie: dwie ścieżki wskazujące na ten sam plik
- usunięcie kontenera nie usuwa zasobu hosta
- co gdy w obu zasobach aFile.txt ?
  - host wygrywa
  - ale nie nadpisuje
  - kontener bez bind mount - pojawi się aFile.txt kontenera

bind mounting

- konfiguracja
  - **nie** można w **Dockerfile**
  - docker **container run** -v
    - podobnie jak named volumes, zasób zamiast nazwy
    - … run -v /Users/luka/zasob:/container/path (mac/ linux)
    - … run -v //c/Users/luka/zasob:/container/path (win)

Dockerfile & Spring Boot

TODO

- utwórz folder docker_dockerfile, w folderze
  - projekt Spring Boot (Web, ”Hello Docker”)
  - Dockerfile

Dockerfile & Spring Boot

Spring Boot Web

Spring Boot Web:

- @GetMapping(”/”)
- zwraca ”Hello Docker”

Dockerfile & Spring Boot

Dockerfile

```
# budujemy z alpine
 FROM openjdk:8-jdk-alpine
# dla Tomcata
VOLUME /tmp
# kopiowanie z target do obrazu
COPY ./<projekt>/target/<nazwa>.jar  /opt/app.jar
# uruchomienie jar’a
ENTRYPOINT [“java","-Djava.security.egd=file:/dev/./urandom","-jar","/opt/app.jar"]
```


Dockerfile & Spring Boot

Uruchomienie

- docker_dockerfile
  - zbudować mavenem projekt boot
  - docker build . -t my-dockerized-boot
  - docker run -p 8080:8080 my-dockerized-boot
- http://localhost:8080

Docker Compose: dlaczego?

- **konfiguracja zależności** pomiędzy kontenerami
- zapisanie konfiguracji **w jednym pliku**
- ‘**one-liner**’ dla postawienia rozproszonego środ.
- na produkcji - Docker Swarm, Kubernetes etc.
- Swarm vs Kubernetes
  - https://platform9.com/blog/kubernetes-docker-swarm-compared/

Docker Compose - co to takiego?

- **plik konfiguracyjny** docker-compose.yml
  - opcje dla kontenerów, sieci, itd.
- **CLI docker-compose**
  - narzędzie dla automatyzacji, używa pliku config.

docker-compose.yml

- **format YAML**, pierwsza linia to wersja pliku
  - 1, 2, 2.1, 3, 3.1
  - z czasem plik konfiguracyjny dojrzewał, wyższa wersja pozwala więcej konfigurować
  - jeśli nie oidamy wersji, domyślna v1 - **rekomendowana: minimum v2**
- używany z **docker-compose lokalnie** (test / dev)
- używany z **docker produkcyjnie**
  - od v1.13, ze Swarm
- **docker-compose.yml** to **domyślna** nazwa, możemy **własną**
  - wtedy trzeba dodać flagę: docker-compose **-f**

docker-compose CLI

- do lokalnej pracy
- docker-compose up # start
- docker-compose down # clean up
- przykładowe zapoznanie nowej osoby z projektem:
  - git clone github.com/repo/projektu
  - docker-compose up

docker-compose.yml & Spring Boot
TODO

- utwórz folder docker_compose
- w docker_compose utwórz / skopiuj z docker_dockerfile
- Dockerfile
- docker-compose.yml
- projekt Spring Boot Web

docker-compose.yml & Spring Boot

Spring Boot Web

skopiuj z docker_dockerfile :)

docker-compose.yml & Spring Boot

Dockerfile

Dockerfile

FROM openjdk:8-jdk-alpine

COPY ./<nazwa_projektu>/target/<nazwa>.jar /opy/app.jar

ENTRYPOINT [“java”,"-Djava.security.egd=file:/dev/./urandom","-jar","/opt/app.jar"]

docker-compose.yml & Spring Boot

docker-compose.yml

docker-compose.yml

```
version: '3'
services:
  boot-project:
    image: boot-image
    build: .
    ports:
      - ‘8080:8080'
    volumes:
      - boot-project-tmp:/tmp
volumes:
  boot-project-tmp:
```

docker-compose.yml & Spring Boot

uruchomienie

docker-compose up