PGS SOFTWARE
Part of Xebia

Techniki malarskie w dockerze

Wszystko, co chcielibyście wiedzieć o obrazach, ale baliście się zapytać

Michał Borkowski

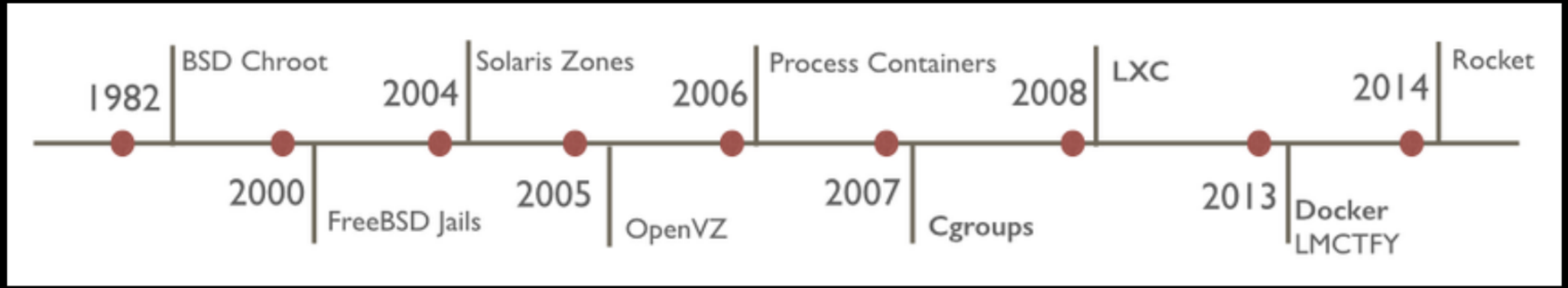
mborkowski@pgs-soft.com

https://norasoft.eu/talks/docker-painting-techniques/

O czym dzisiaj?

- historia obrazów
- czym jest obraz?
- jak namalować obraz?
- analiza artystyczna
- jak nie malować obrazu?

Nieznana historia malarstwa



Docker jest wszędzie

Google and Containers

Everything at Google runs in a container.

Internal usage:

- Resource isolation and predictability
- Quality of Services
  - batch vs. latency sensitive serving
- Overcommitment (not for GCE)
- Resource Accounting

We start over 2 billion containers per week.

Google Cloud Platform

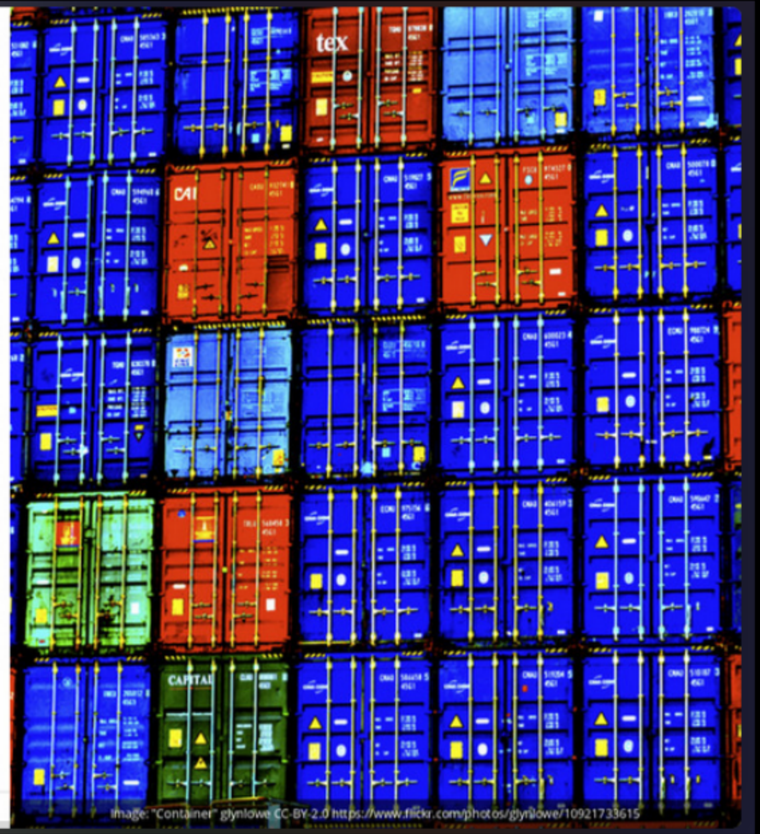
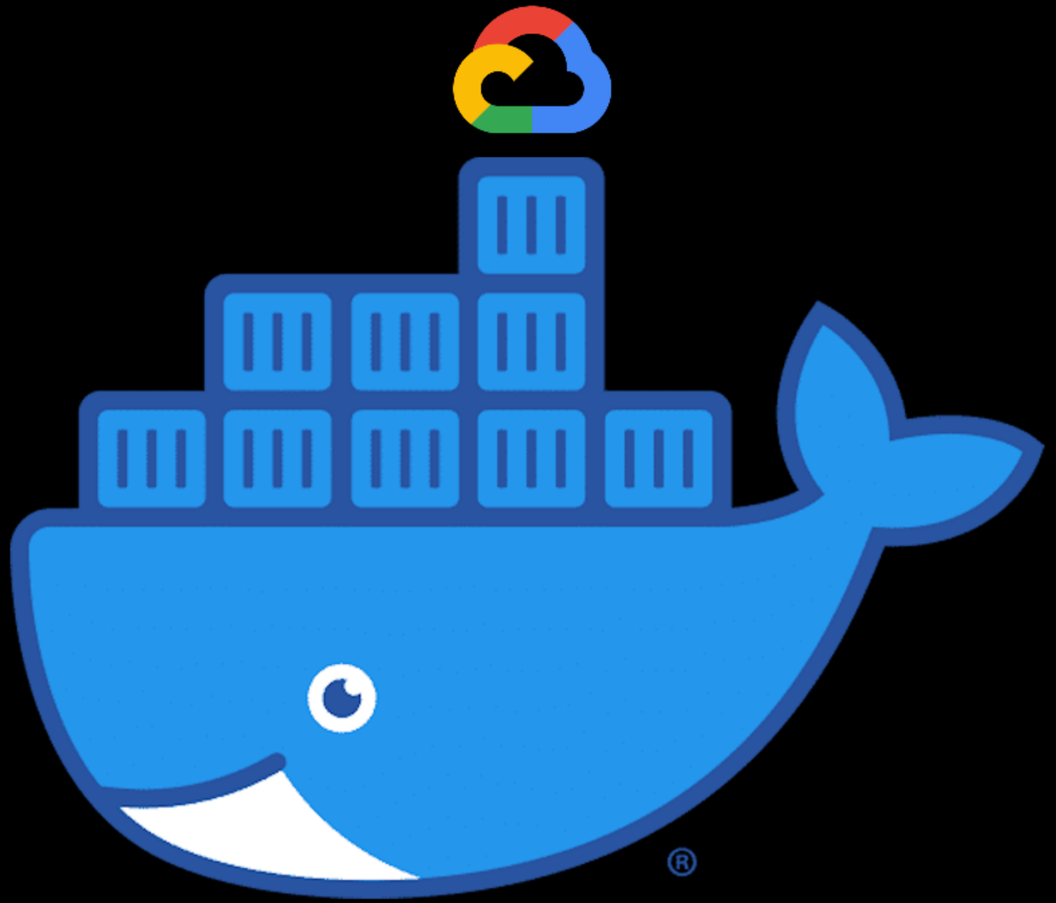


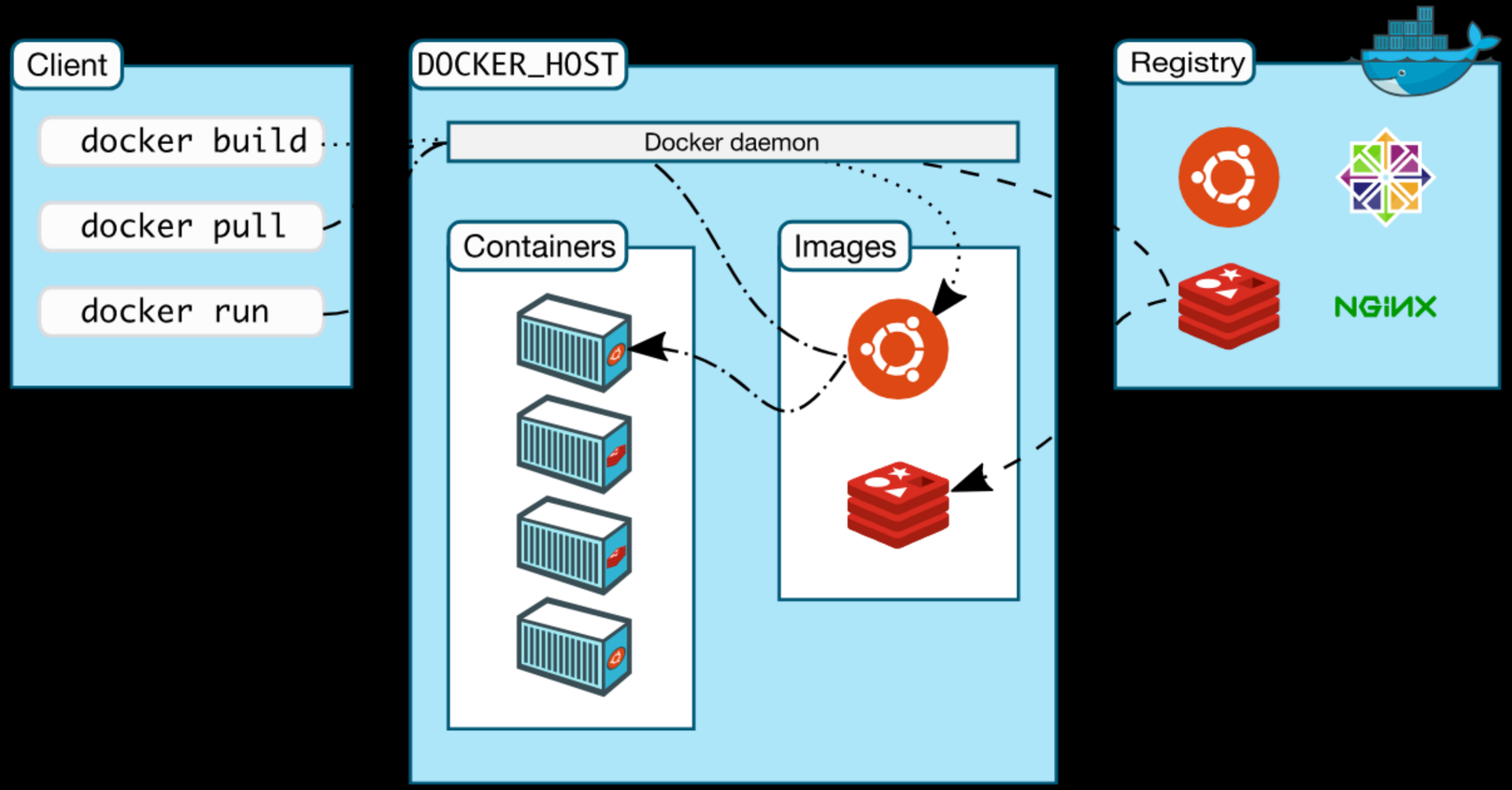
image: "Container" glynlowe CC-BY-2.0 https://www.flickr.com/photos/glynlowe/10921733615

Docker jest wszędzie

Docker jest wszędzie



Komponenty dockera



From docker documentation

RunC - jak zbudować kontener?

```
runc-container
├── config.json
└── rootfs
    ├── bin -> usr/bin
    ├── boot
    ├── dev
    ├── etc
    ├── home
    ├── lib -> usr/lib
    ├── lib32 -> usr/lib32
    ├── lib64 -> usr/lib64
    ├── libx32 -> usr/libx32
    ├── media
    ├── mnt
    ├── opt
    ├── proc
```

```
{
    "ociVersion": "1.0.2-dev",
    "process": {
        "terminal": true,
        "user": {
            "uid": 1001,
            "gid": 1001
        },
        "args": [
            "/bin/bash"
        ],
        "env": [
            "PATH=/usr/local/sbin:/usr/local
            "TERM=xterm"
        ],
```

Czym jest obraz?

Obrazy mają warstwy

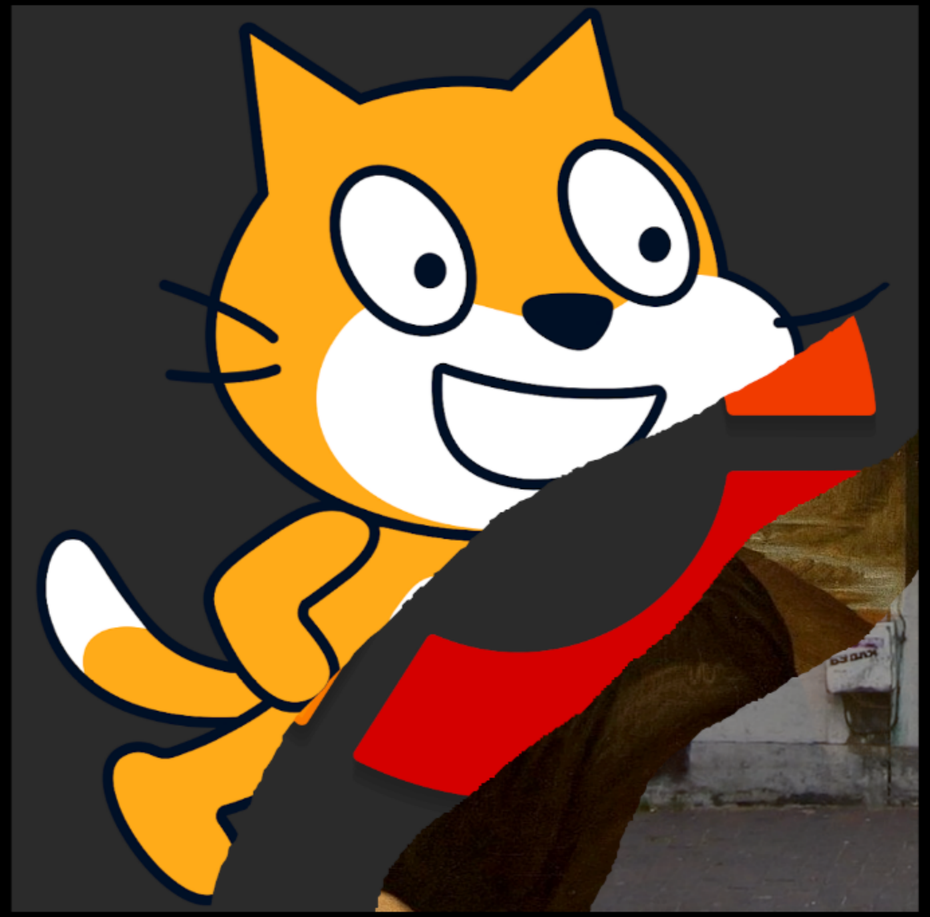

msc
msc
msc
msc

Obrazy... aż do końca

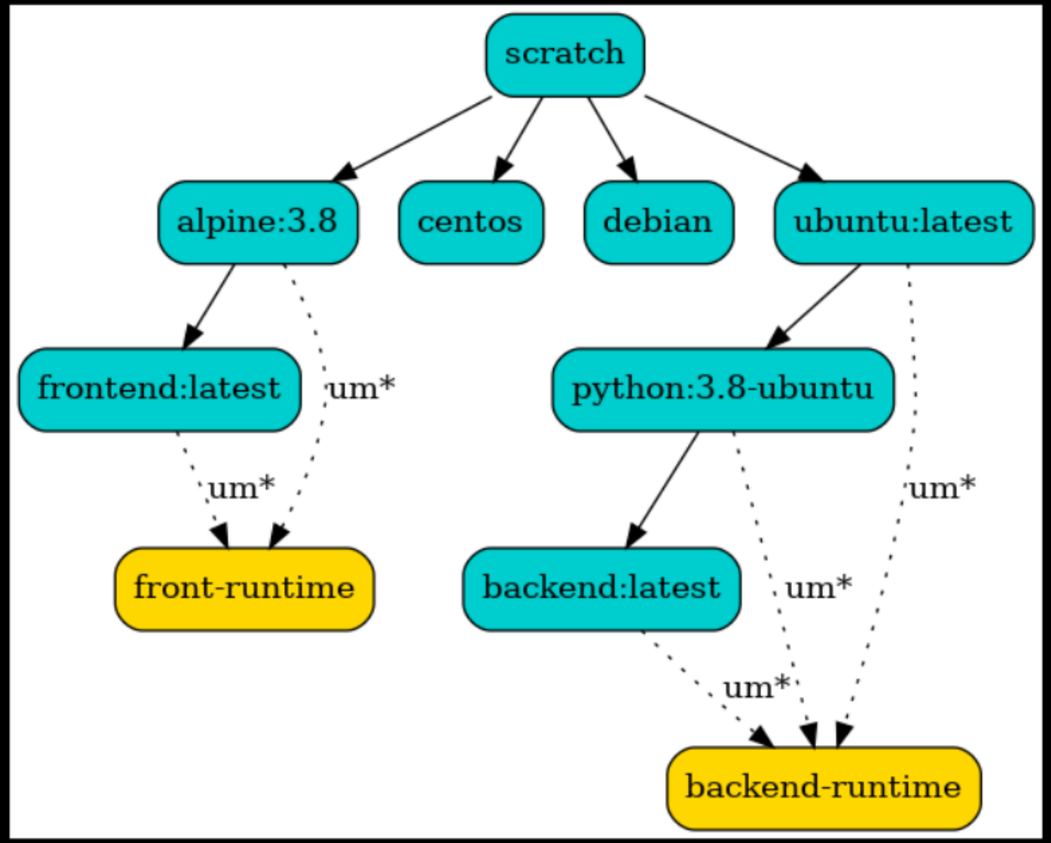
Obrazy... aż do końca

Obrazy... aż do końca

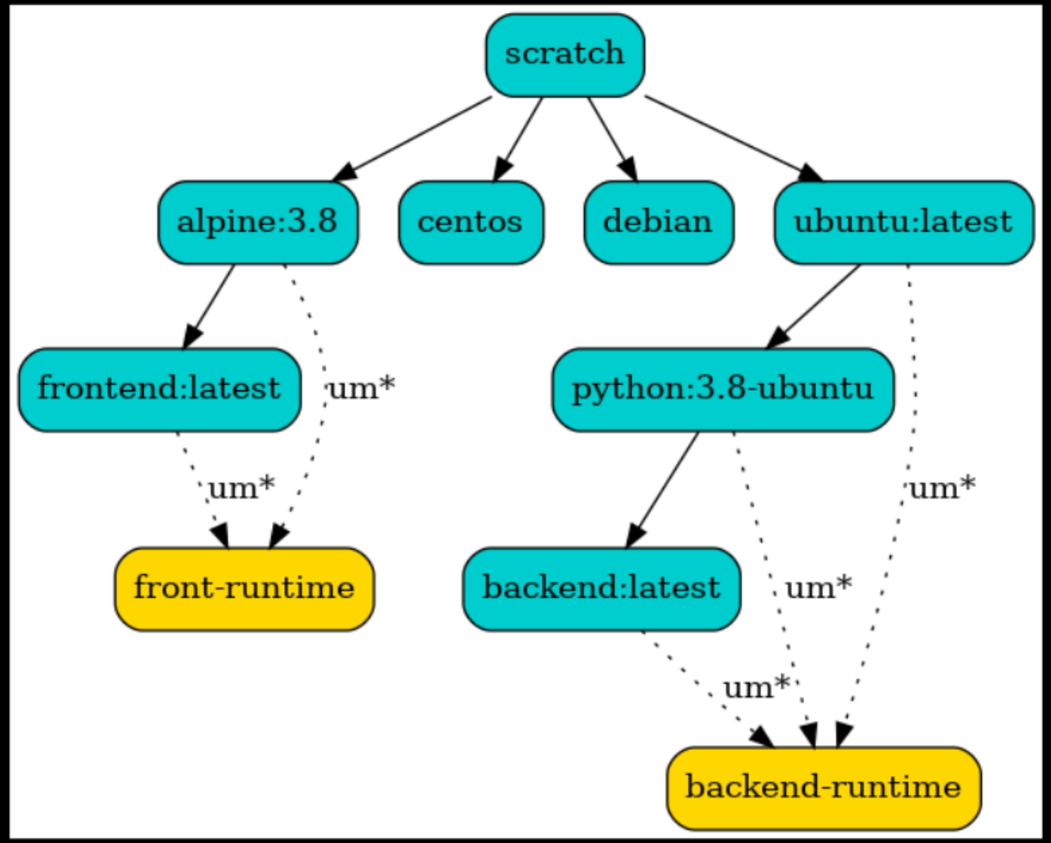
Obrazy... aż do końca



Dobrze znane warstwy

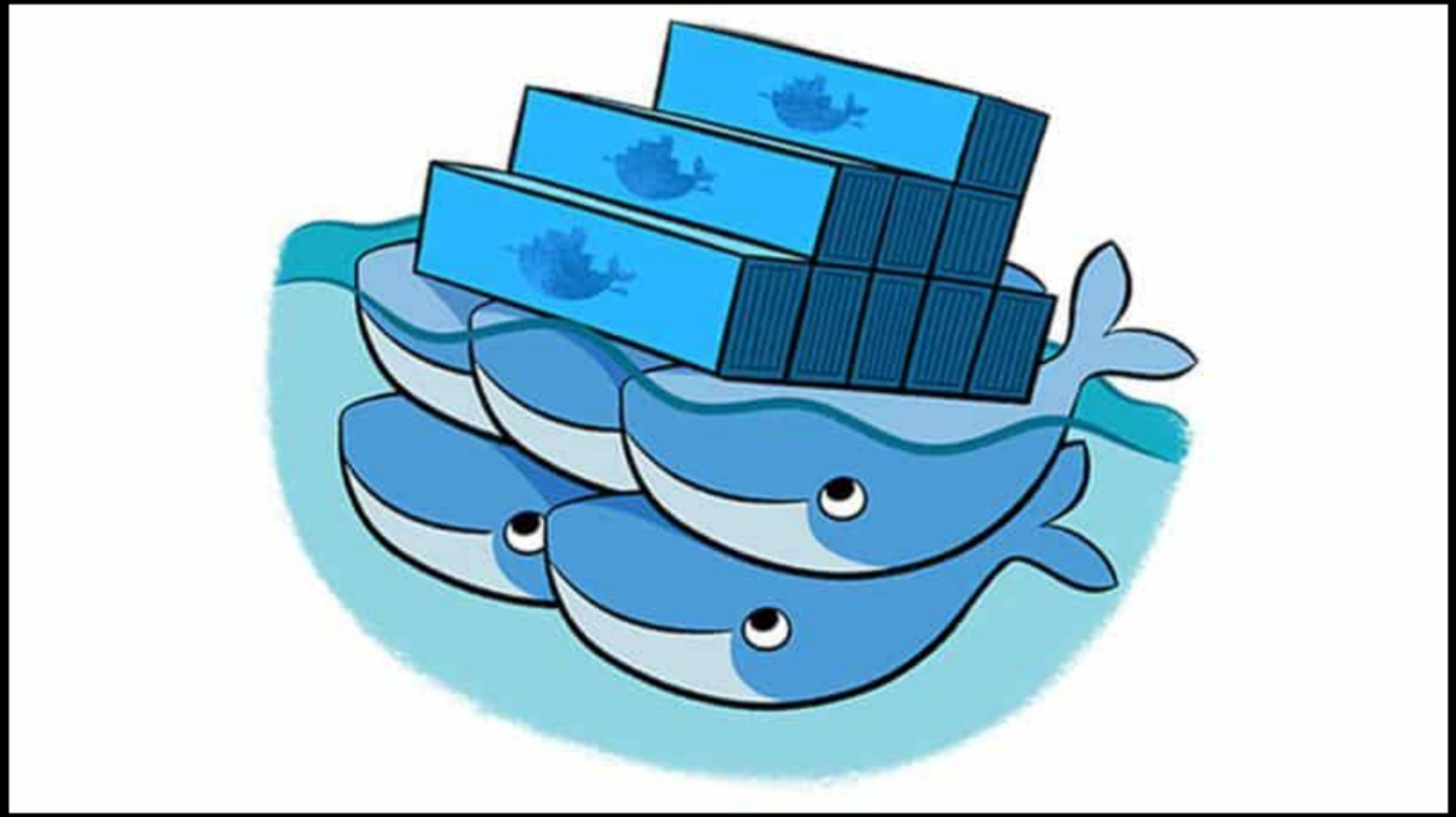


Dobrze znane warstwy



um - Union Mount

Trzymane na cudzych komputerach



Jak namalować obraz?

Weź gotowy podkład

```
FROM ubuntu:22.04
```

I dodaj kolejne warstwy

```
FROM ubuntu:22.04

WORKDIR /app

COPY hello.py ./

RUN apt-get update
RUN apt-get install python3 -y

CMD [ "python3", "hello.py" ]
```

Poczekaj aż farba wyschnie

```
docker build -t python-hello .
```

Oceń efekt

Oceń efekt

```
docker inspect python-hello
```

```
cat /var/lib/docker/image/overlay2/imagedb/content/sha256/<image-id>
```

```
zcat /var/lib/docker/image/overlay2/layerdb/sha256/<layer-id>/tar-split.json.gz
```

Oceń efekt

```
dive python-hello
```

Jak nie malować obrazu?

Dlaczego ten obraz jest taki duży?

```
FROM ubuntu:22.04

WORKDIR /app

COPY hello.py ./

RUN apt-get update
RUN apt-get install python3 -y

CMD [ "python3", "hello.py" ]
```

Dlaczego ten obraz jest taki duży?

```
FROM ubuntu:22.04

WORKDIR /app

COPY hello.py ./

RUN apt-get update
RUN apt-get install python3 -y
RUN rm -rf /var/lib/apt/lists/*

CMD [ "python3", "hello.py" ]
```

Dlaczego ten obraz jest taki duży?

```
FROM ubuntu:22.04

WORKDIR /app

COPY hello.py ./

RUN apt-get update && \
  apt-get install python3 -y && \
  rm -rf /var/lib/apt/lists/*

CMD [ "python3", "hello.py" ]
```

Dlaczego ten obraz jest taki duży?

```
RUN apt-get update && \
  apt-get install -y python3 python3-pip gcc libsasl2-dev \
  python-dev libldap2-dev libssl-dev  && \
  pip install -r requirements.txt && \
  apt-get purge -y python3-pip gcc libsasl2-dev python-dev \
  libldap2-dev libssl-dev && \
  rm -rf /var/lib/apt/lists/*
```

To może inna technologia

```
FROM golang:latest

WORKDIR /app

COPY hello.go ./

CMD [ "go", "run", "hello.go" ]
```

Dlaczego obrazy tak długo się budują?

Dlaczego obrazy tak długo się budują?

```
FROM ubuntu:22.04

WORKDIR /app

COPY hello.py ./

RUN apt-get update && apt-get install -y python3

CMD [ "python3", "hello.py" ]
```

Dlaczego obrazy tak długo się budują?

```
FROM ubuntu:22.04 as builder1
RUN apt-get update && apt-get install -y golang
COPY source1.go source.go
RUN go build -o /binary source.go

FROM ubuntu:22.04 as builder2
RUN apt-get update && apt-get install -y golang
COPY source2.go source.go
RUN go build -o /binary source.go

FROM ubuntu:22.04
COPY --from=builder1 /binary /binary1
COPY --from=builder2 /binary /binary2
CMD [ "/binary1", "||", "/binary2" ]
```

Dlaczego obrazy tak długo się budują?

```
FROM ubuntu:22.04 as shared
RUN apt-get update && apt-get install -y golang

FROM shared as builder1
COPY source1.go source.go
RUN go build -o /binary source.go

FROM shared as builder2
COPY source2.go source.go
RUN go build -o /binary source.go

FROM ubuntu:22.04
COPY --from=builder1 /binary /binary1
COPY --from=builder2 /binary /binary2
CMD [ "/binary1", "||", "/binary2" ]
```

Dlaczego obrazy tak długo się budują?

```
docker build --target shared -t go-builder-shared .

docker build -t go-hello-multi --cache-from=go-builder-shared .
```

Docker na sterydach

```
export DOCKER_BUILDKIT=1
docker build -t go-hello-multi --cache-from=go-builder-shared .
```

```
go-hello-multi > DOCKER_BUILDKIT=1 docker build --no-cache -t go-hello-multi -f Dockerfile.shared .          ~prez/docker-image-examples/go-hello-multi
[+] Building 41.0s (13/13) FINISHED
 => [internal] load build definition from Dockerfile.shared                                                          0.0s
 => => transferring dockerfile: 440B                                                                                 0.0s
 => [internal] load .dockerignore                                                                                    0.0s
 => => transferring context: 2B                                                                                      0.0s
 => [internal] load metadata for docker.io/library/ubuntu:22.04                                                      0.0s
 => [shared 1/2] FROM docker.io/library/ubuntu:22.04                                                                 0.0s
 => [internal] load build context                                                                                    0.0s
 => => transferring context: 234B                                                                                    0.0s
 => [shared 2/2] RUN apt-get update && apt-get install -y golang                                                    39.6s
 => [builder1 1/2] COPY source1.go source.go                                                                         0.0s
 => [builder2 1/2] COPY source2.go source.go                                                                         0.0s
 => [builder2 2/2] RUN go build -o /binary source.go                                                                 0.8s
 => [builder1 2/2] RUN go build -o /binary source.go                                                                 0.8s
 => [stage-3 2/3] COPY --from=builder1 /binary /binary1                                                              0.0s
 => [stage-3 3/3] COPY --from=builder2 /binary /binary2                                                              0.0s
 => exporting to image                                                                                               0.0s
 => => exporting layers                                                                                              0.0s
 => => writing image sha256:8411f8e426e378f65028eb759d860c5fa4c1fff7eba27924e6e27c7338dcf2d3                         0.0s
 => => naming to docker.io/library/go-hello-multi                                                                    0.0s
go-hello-multi >                                                                                                    ~prez/docker-image-examples/go-hello-multi
```

Jak wyciekły nasze hasła?

Jak wyciekły nasze hasła?

```
FROM ubuntu:22.04

WORKDIR /app

RUN echo pip install --extra-index-url \
  "https://user:password@nexus.company.com/repository" \
  -r requirements > /app/output.txt

CMD [ "cat", "output.txt" ]
```

```
docker build -t secure-hello .
```

Jak wyciekły nasze hasła?

```
FROM ubuntu:22.04

WORKDIR /app

ARG NEXUS_USER
ARG NEXUS_PASS

RUN echo pip install --extra-index-url \
  "https://${NEXUS_USER}:${NEXUS_PASS}@nexus.company.com/repository" \
  -r requirements > /app/output.txt

CMD [ "cat", "output.txt" ]
```

```
docker build --build-arg NEXUS_USER=user --build-arg NEXUS_PASS=pass -t secure-pass .
```

Jak wyciekły nasze hasła?

```
FROM ubuntu:22.04

WORKDIR /app

RUN --mount=type=secret,id=netrc,uid=0 \
  NETRC=/run/secrets/netrc echo pip install \
  --extra-index-url "https://nexus.company.com/repository" \
  -r requirements > /app/output && \
  cp /run/secrets/netrc /app/netrc

CMD [ "cat", "output", "netrc" ]
```

```
export DOCKER_BUILDKIT=1
export CUSTOM_NETRC="machine nexus.company.com
login user
password pass
"
docker build --secret id=netrc,env=CUSTOM_NETRC -t secure-pass .
```

A może wyciekł klucz prywatny

```
FROM ubuntu:22.04

WORKDIR /app

RUN mkdir -p ~/.ssh
COPY ~/.ssh/id_rsa ~/.ssh/id_rsa
RUN echo "Host *.trabe.io\n\tStrictHostKeyChecking no\n" >> ~/.ssh/config
RUN git clone git@github.com:awesome-author/my-shared-repo.git
RUN rm -rf ~/.ssh/id_rsa
```

A może wyciekł klucz prywatny

```
# ustaw domyślny klucz SSH
ssh-add ~/.ssh/id_rsa

# albo użyj konkretnego w czasie wykonania
DOCKER_BUILDKIT=1 docker build --ssh default=$HOME/.ssh/my_custom_key_rsa .
```

```
FROM ubuntu:22.04

WORKDIR /app

RUN mkdir -p /root/.ssh && ssh-keyscan github.com >> ~/.ssh/known_hosts
RUN --mount=type=ssh git clone git@github.com:awesome-author/my-shared-repo.git
```

Docker compose też tak potrafi :)

```
docker compose build --ssh default=$HOME/.ssh/my_custom_key_rsa
```

PGS SOFTWARE
Part of Xebia

https://norasoft.eu/talks/docker-painting-techniques/

mborkowski@pgs-soft.com