



# JAK SKUTECZNIE ZAPROJEKTOWAĆ I WYTWORZYĆ **DOBRY** OBRAZ DOCKEROWY

# BIO

DevOps z **6-letnim doświadczeniem** w branży IT.  
Na co dzień zajmuje się utrzymaniem infrastruktury on-prem jak i cloudowej.  
Interesuje się **automatyzacją** oraz bezpieczeństwem.



# PLAN WARSZTATU

- Omówienie budowy logicznej obrazów dockerowych – niestety musi być ☺
- Przydatne narzędzia do projektowania obrazów dockerowych
- Jak zaprojektować warstwy dockerowe?
- Nasz przyjaciel – System CI/CD

Przerwy co około 1h po 15m






## LINK DO MATERIAŁÓW

- <https://github.com/ByJacob/stacjait-szkolenie-docker-image>



# CO MOŻNA ZNALEŹĆ W INTERNECIE?



```
FROM ubuntu:latest  
COPY /source /destination  
RUN touch /example-demo
```

# CO MOŻNA ZNALEŹĆ W INTERNECIE?

```
# syntax=docker/dockerfile:1
FROM node:18-alpine
WORKDIR /app
COPY . .
RUN yarn install --production
CMD ["node", "src/index.js"]
EXPOSE 3000
```

# CO MOŻNA ZNALEŹĆ W INTERNECIE?

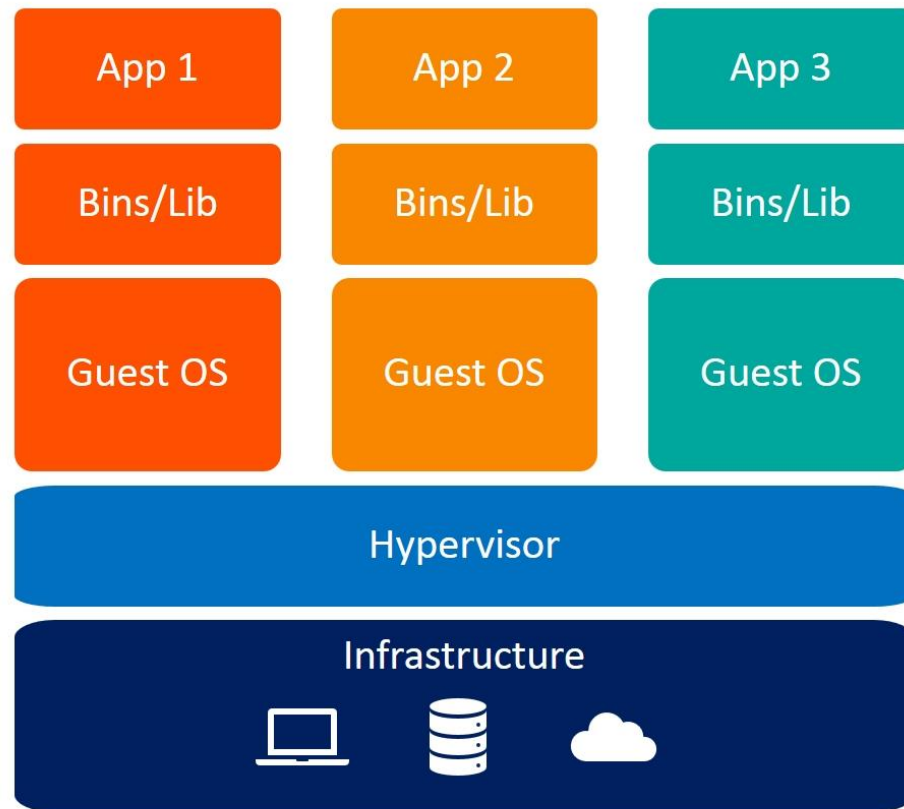
```
1 FROM python:3.9.1-buster
2
3 COPY files/smallfile.txt /tmp/files/
4
5 RUN pip install pip --upgrade && \
6     pip install \
7         ansible \
8         awscli
9
10 # Another intensive workload (eg. building bin from source)
11 RUN sleep 30
12
13 ENTRYPOINT ["/bin/bash"]
```

# CO MOŻNA ZNALEŹĆ W INTERNECIE?

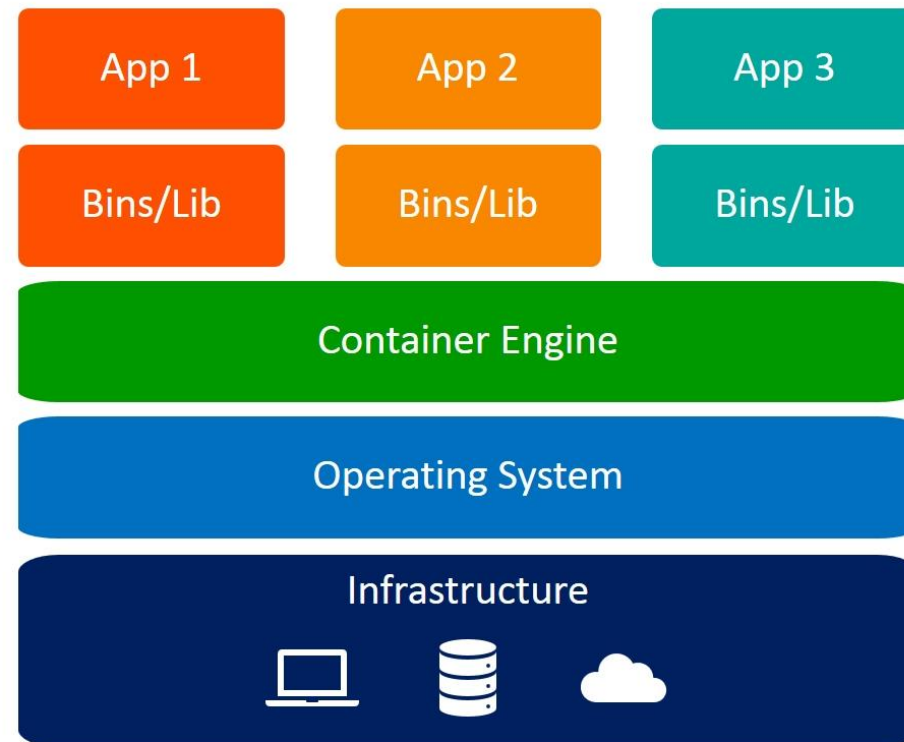
```
#This is the "builder" stage
FROM golang:1.15 as builder
WORKDIR /my-go-app
COPY app-src .
RUN GOOS=linux GOARCH=amd64 go build ./cmd/app-service
#This is the final stage, and we copy artifacts from "builder"
FROM gcr.io/distroless/static-debian10
COPY --from=builder /my-go-app/app-service /bin/app-service
ENTRYPOINT ["/bin/app-service"]
```



# DOCKER VS VM

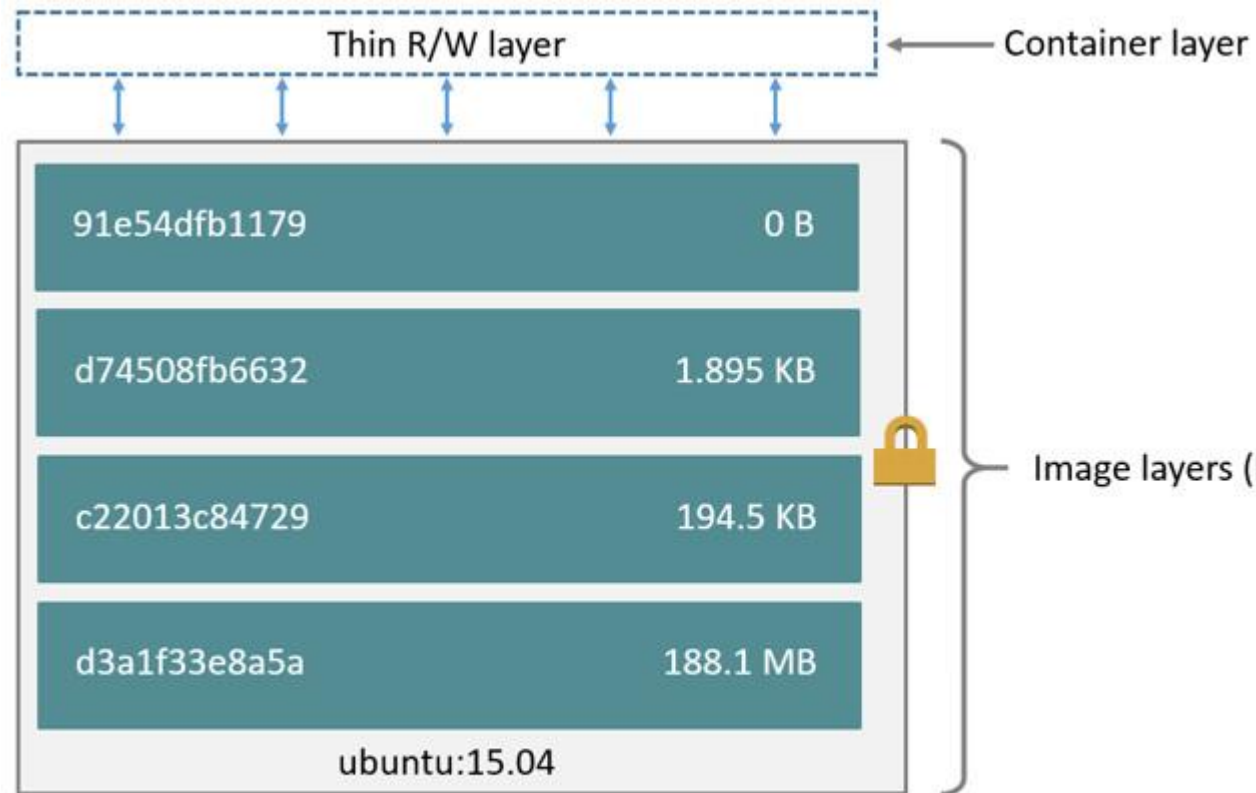


Virtual Machines



Containers

# WYJAŚNIENIE WARSTW W DOKERZE



Container  
(based on ubuntu:15.04 image)

Tworzenie nowego obrazu

```
RUN rm -rf /usr/share/nginx/bardzo_duzy_plik.tar.gz
```

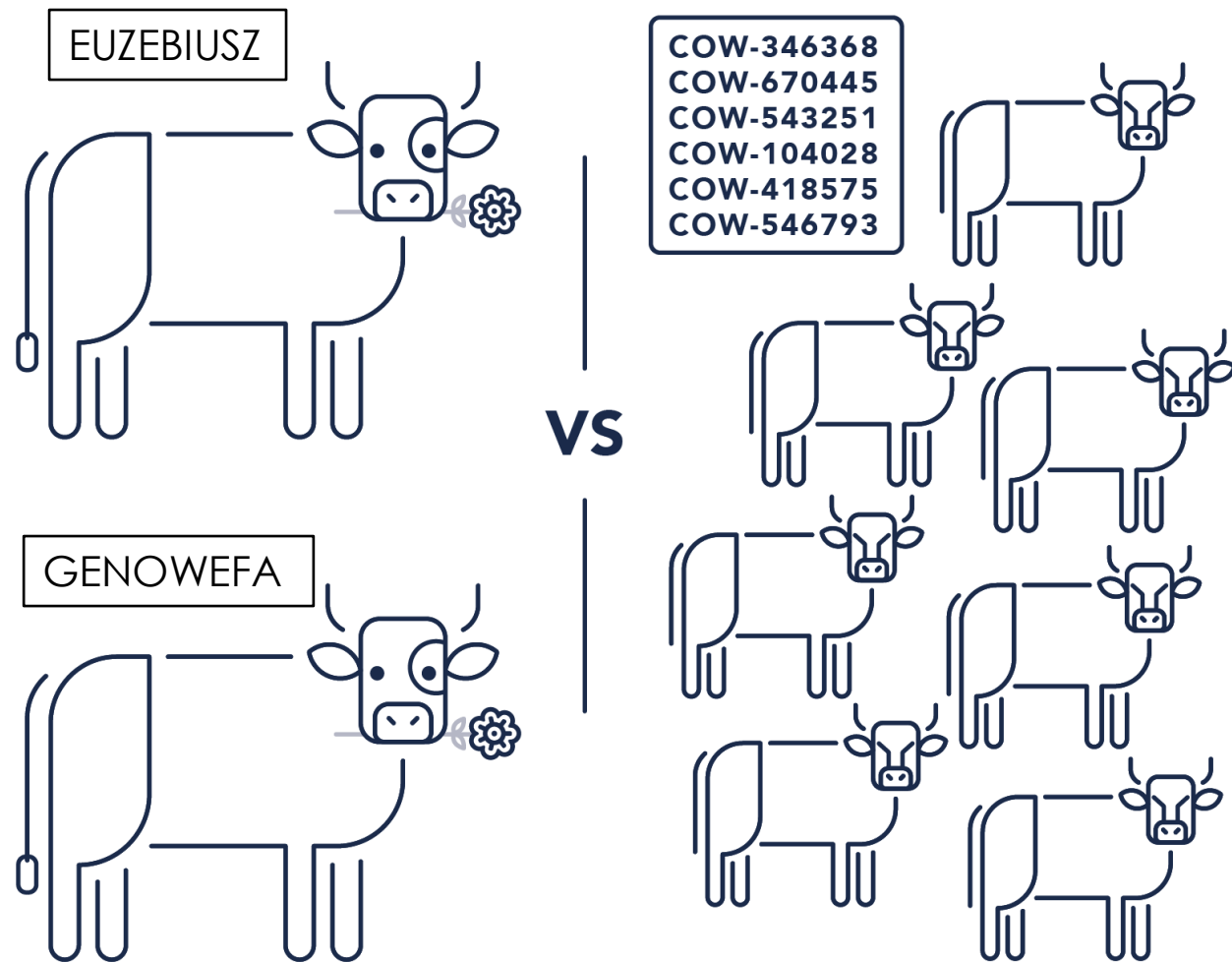
```
COPY . /usr/share/nginx/
```

```
RUN apt-get update && apt-get install -y vim
```

```
FROM nginx
```

# EFEMERYCZNOŚĆ KONTENERÓW

Podejście  
Pets vs. Cattle  
(zwierzątka vs bydło)





# DEMO1

# DODATKOWE INFORMACJE O WARSZTACIE

A piggy bank and a pile of coins. The piggy bank is a light brown color and is partially filled with coins. The coins are of various denominations, including 1, 2, and 5 zł. The background is a dark gray gradient.

- Warsztaty te nie dadzą Państwu „złotego leku” na problemy w prowadzonych projektach.
- Mają jedynie za zadanie pokazać dalsze możliwości rozwoju i optymalizacji skonteneryzowanej aplikacji.

**ADAPTUJ, nie ADOPTUJ**



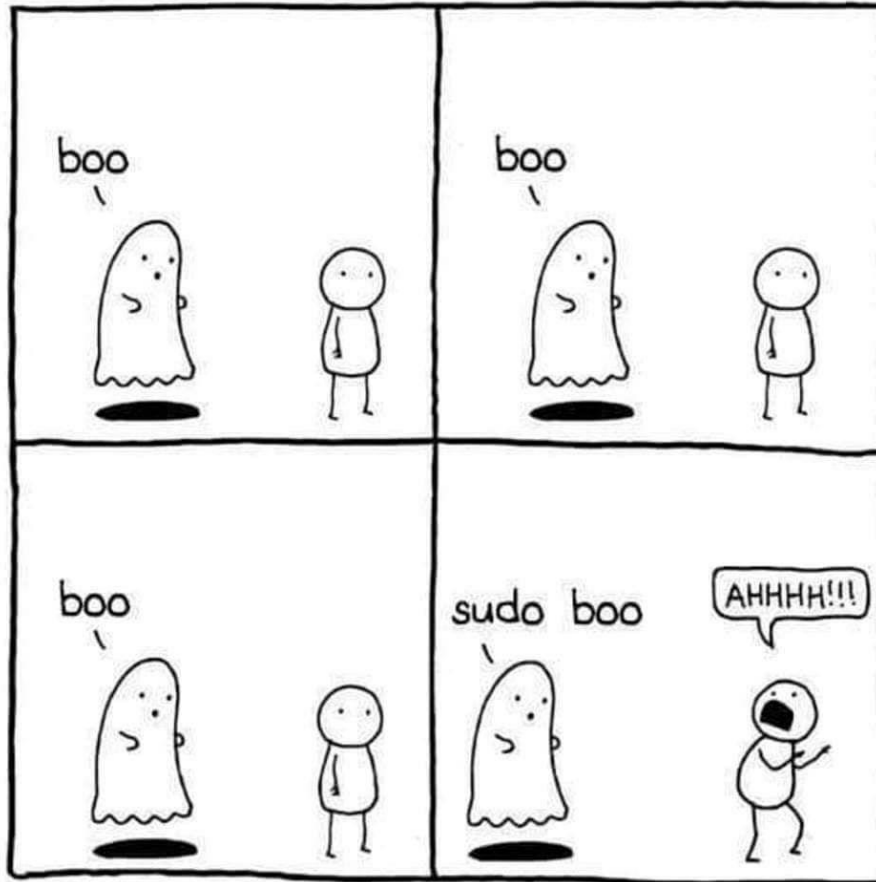
The background is a technical drawing or blueprint. It features various geometric shapes, circles, and lines. A yellow pencil is positioned diagonally in the upper left. A large, detailed drawing of a ball bearing is in the center. To the right, a vernier caliper is shown with its jaws open. The text is overlaid on the left side of the image.

# PRZYDATNE NARZĘDZIA DO PROJEKTOWANIA OBRAZÓW DOCKEROWYCH

# HADOLINT

- <https://github.com/hadolint/hadolint>
- Linter weryfikujący najlepsze praktyki pisanie Dockerfile
- Dostępny w wersji online i CLI

```
DL4000 Specify a maintainer of the Dockerfile
DL3006 Always tag the version of an image explicitly.
1 FROM debian
SC1007 Remove space after = if trying to assign a value (for empty string, use var='' ... ).
SC2154 node_version is referenced but not assigned.
DL3009 Delete the apt-get lists after installing something
2 RUN node_version= "0.10" \
3   && apt-get update && apt-get -y install nodejs="$node_version"
4 COPY package.json usr/src/app
DL3003 Use WORKDIR to switch to a directory
5 RUN cd /usr/src/app \
6   && npm install node-static
7
DL3011 Valid UNIX ports range from 0 to 65535
8 EXPOSE 80000
9 CMD ["npm", "start"]
```



DEMO2

# DIVE

- <https://github.com/wagoodman/dive>
- Narzędzie do „eksploracji” warstw obrazu dockerowego
- **Dodatkowo:** analiza marnowania miejsca

```
[wagoodman@kiwi dive] ❯ ci-integration $ CI=true build/dive dive-test
Using config file: /home/wagoodman/.dive.yaml
Fetching image...
Parsing image...
┌ [layer: 1] 1871059774abe69 : [=====] 100 % (1/1)
┌ [layer: 2] 28cfe03618aa2e9 : [=====] 100 % (415/415)
┌ [layer: 3] 3d4ad907517a021 : [=====] 100 % (2/2)
┌ [layer: 4] 461885fc2258915 : [=====] 100 % (4/4)
┌ [layer: 5] 49fe2a475548bfa : [=====] 100 % (4/4)
┌ [layer: 6] 5eca617bdc3bc06 : [=====] 100 % (3/3)
┌ [layer: 7] 80cd2ca1ffc8996 : [=====] 100 % (3/3)
┌ [layer: 8] 81b1b002d4b4c13 : [=====] 100 % (3/3)
┌ [layer: 9] a10327f68ffed4a : [=====] 100 % (2/2)
┌ [layer: 10] aad36d0b05e71c7 : [=====] 100 % (4/4)
┌ [layer: 11] c99e2f8d3f62826 : [=====] 100 % (3/3)
┌ [layer: 12] cfb35bb5c127d84 : [=====] 100 % (2/2)
┌ [layer: 13] f07c3eb88757239 : [=====] 100 % (3/3)
└ [layer: 14] f2fc54e25cb7966 : [=====] 100 % (2/2)
=
Analyzing image...
  efficiency: 98.4421 %
  wastedBytes: 32025 bytes (32 kB)
  userWastedPercent: 2.6376 %
Run CI Validations...
  Using CI config: .dive-ci
  PASS: highestUserWastedPercent
  SKIP: highestWastedBytes: rule disabled
  FAIL: lowestEfficiency: image efficiency is too low (efficiency=0.9844212134184309 < threshold=0.99)
Result:FAIL [Total:3] [Passed:1] [Failed:1] [Warn:0] [Skipped:1]
X:1 [wagoodman@kiwi dive] ❯ ci-integration $
```



## How Linux introducing New Update



& Windows New Update

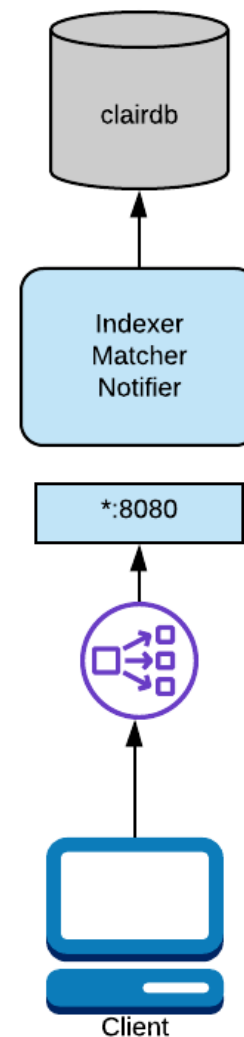


# DEMO3



# CLAIR

- <https://github.com/quay/clair>
- Projekt open source do statycznej analizy luk w kontenerach aplikacji
- **Indexer** - próbuje odkryć, jakie pakiety istnieją w obrazie, z jakiej dystrybucji pochodzi obraz i jakie repozytoria pakietów są używane w obrazie
- **Matcher** - domyślnie aktualizuje bazę podatności i dostarcza raport podatności po zapytaniu
- **Notifier** - śledzi nowe aktualizacje i w razie jakichkolwiek zmian odnośnie zaindeksowanych elementów może wysłać powiadomienie
- [https://quay.github.io/claircore/concepts/severity\\_mapping.html](https://quay.github.io/claircore/concepts/severity_mapping.html)



# DOCKER SCAN

- <https://docs.docker.com/engine/scan/>
- Lokalne skanowanie obrazów dockerowych pod kątem luk w zabezpieczeniach
- Limit 10 lub 200 skanowań miesięcznie po zalogowaniu
- **Wymaga zalogowania do dockerhub**

```
$ docker scan --accept-license --version
Version:      v0.12.0
Git commit:   1074dd0
Provider:     Snyk (1.790.0 (standalone))
```

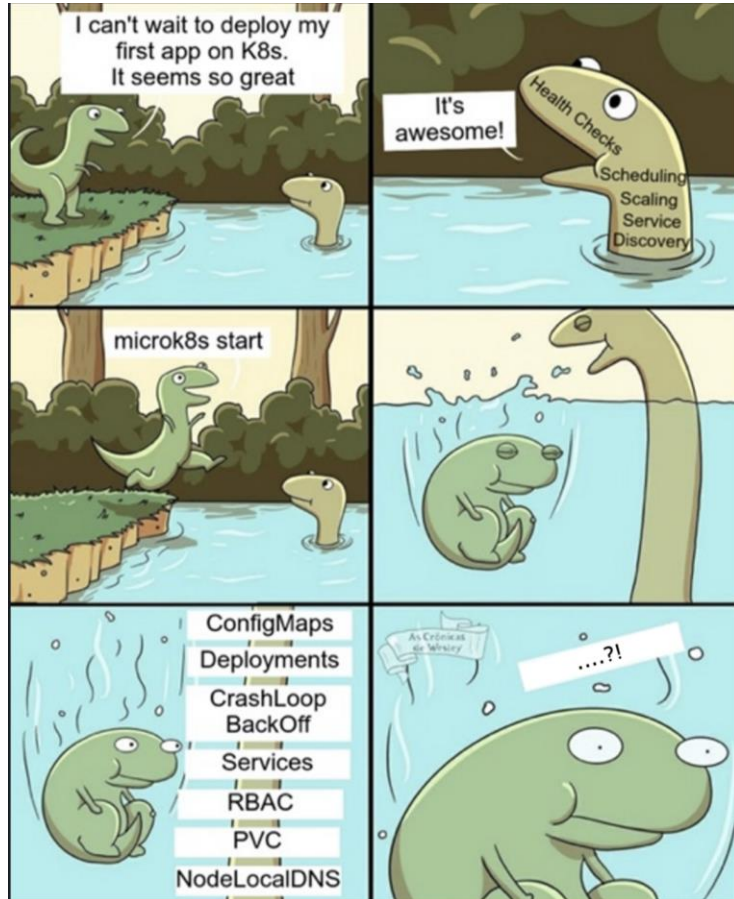
```
$ docker scan hello-world

Testing hello-world...

Organization:    docker-desktop-test
Package manager: linux
Project name:    docker-image|hello-world
Docker image:    hello-world
Licenses:        enabled

✓ Tested 0 dependencies for known issues, no vulnerable paths found.

Note that we do not currently have vulnerability data for your image.
```



# DEMO4

# S6-OVERLAY

- <https://github.com/just-containers/s6-overlay>
- <https://skarnet.org/software/s6/overview.html>
- Zbiór narzędzi do nadzorowania, zarządzania, logowania i inicjalizacji kontenera
- Utrzymuje jeden proces główny w kontenerze, jednocześnie pozwalając na uruchomienie i monitorowanie wielu procesów potomnych

```
docker-host $ docker build -t demo .
docker-host $ docker run --name s6demo -d -p 80:80 demo
docker-host $ docker top s6demo acxf
```

PID	TTY	STAT	TIME	COMMAND
11735	?	Ss	0:00	\_ s6-svscan
11772	?	S	0:00	\_ s6-supervise
11773	?	Ss	0:00	\_ s6-linux-init-s
11771	?	Ss	0:00	\_ rc.init
11812	?	S	0:00	\_ nginx
11814	?	S	0:00	\_ nginx
11816	?	S	0:00	\_ nginx
11813	?	S	0:00	\_ nginx
11815	?	S	0:00	\_ nginx
11779	?	S	0:00	\_ s6-supervise
11785	?	Ss	0:00	\_ s6-ipcservd
11778	?	S	0:00	\_ s6-supervise

```
docker-host $ curl --head http://127.0.0.1/
HTTP/1.1 200 OK
Server: nginx/1.18.0 (Ubuntu)
Date: Mon, 17 Jan 2022 13:33:58 GMT
Content-Type: text/html
Content-Length: 612
Last-Modified: Mon, 17 Jan 2022 13:32:11 GMT
Connection: keep-alive
ETag: "61e56fdb-264"
Accept-Ranges: bytes
```

Że niby tylko jeden  
proces  
na kontener?



Potrzymaj mi piwo

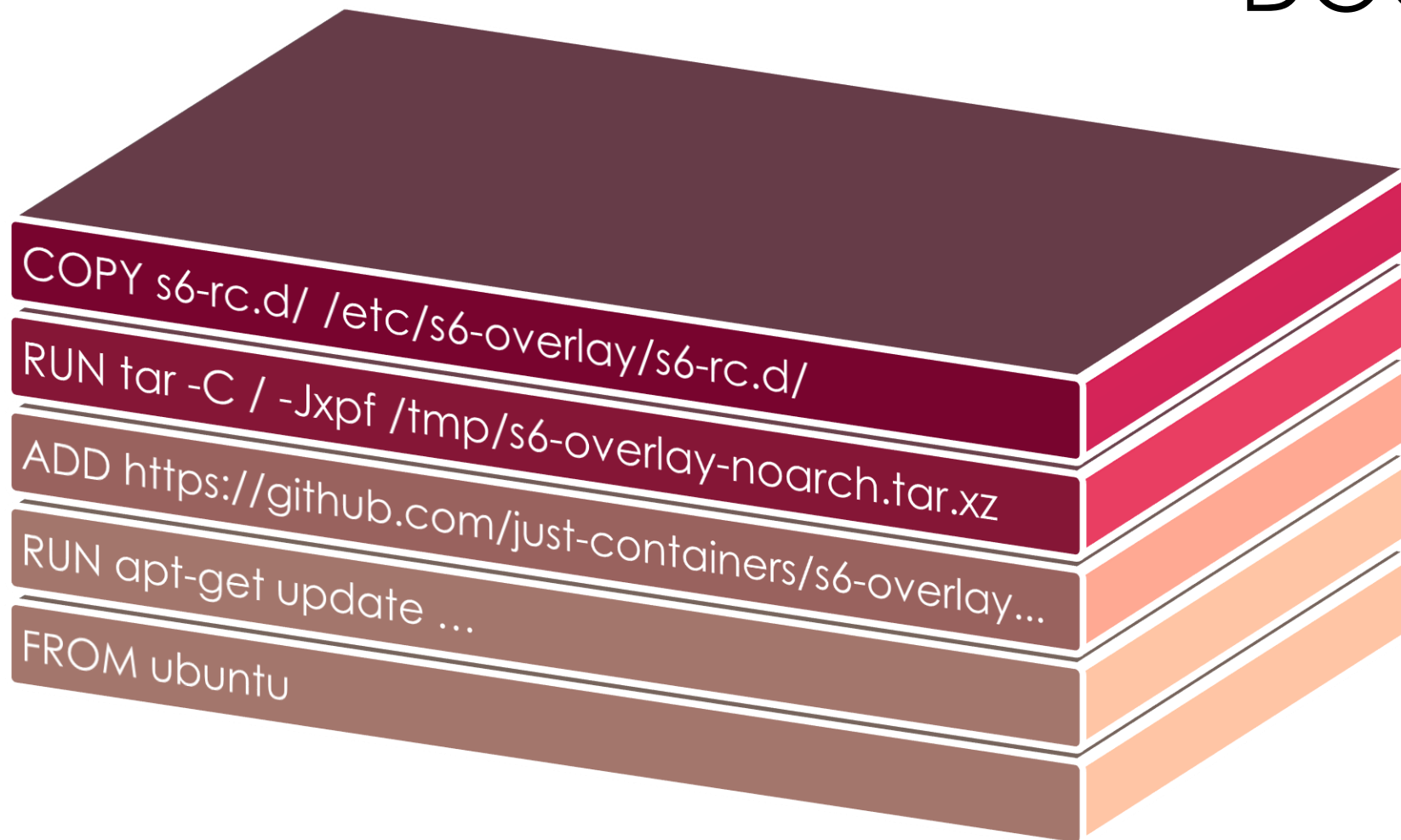
obraz `gitlab/gitlab-ce`

- chef	- redis	...
- crond	- registry	
- gitaly	- spamcheck	
- gitlab	- grafana	
- letsencrypt	- alertmanager	
- logotata	- consul	
- postgresql	- nginx	

# DEMO5



# JAK PROJEKTOWAĆ WARSTWY DOCKEROWE



# HIERARCHIA CZYNNOŚCI PRZY BUDOWANIU OBRAZU

Instalowanie zależności

ŹLE ☹️

```
WORKDIR /my-cool-app
COPY . ./
RUN composer install --no-dev --no-interaction --optimize-autoloader
```

```
FROM python:3.9-slim

WORKDIR /app

COPY . .
RUN pip install -r /requirements.txt
```

DOBRE 😊

```
WORKDIR /my-cool-app
COPY composer.json ./
COPY composer.lock ./
RUN composer install --no-dev --no-interaction --no-autoloader --no-scripts
COPY . ./
RUN composer dump-autoload --optimize
```

```
FROM python:3.9-slim

WORKDIR /app

COPY requirements.txt .

RUN pip install -r /requirements.txt

COPY sample.py .
```

# HIERARCHIA CZYNNOŚCI PRZY BUDOWANIU OBRAZU

Komenda RUN

ŹLE 😞

```
FROM php:8.2-fpm

RUN apt-get update
RUN apt-get install -y --no-install-recommends \
    libfreetype6-dev \
    libicu-dev \
    libjpeg-dev \
    libmagickwand-dev \
    libpng-dev \
    libwebp-dev \
    libzip-dev
RUN docker-php-ext-configure gd \
    --with-freetype \
    --with-jpeg \
    --with-webp
...
RUN rm -rf /var/lib/apt/lists/*
```

DOBRE 😊

```
FROM php:8.2-fpm

RUN apt-get update \
    && apt-get install -y --no-install-recommends \
        libfreetype6-dev \
        libicu-dev \
        libjpeg-dev \
        libmagickwand-dev \
        libpng-dev \
        libwebp-dev \
        libzip-dev \
    && docker-php-ext-configure gd \
        --with-freetype \
        --with-jpeg \
        --with-webp \
    && ... \
    && rm -rf /var/lib/apt/lists/*
```

DOBRE 😊

```
FROM php:8.2-fpm

# -e Exit immediately if a command exits with a non-zero status
# -x Print commands and their arguments as they are executed.
RUN set -ex; \
    \
    apt-get update; \
    apt-get install -y --no-install-recommends \
        libfreetype6-dev \
        libicu-dev \
        libjpeg-dev \
        libmagickwand-dev \
        libpng-dev \
        libwebp-dev \
        libzip-dev; \
    \
    docker-php-ext-configure gd \
        --with-freetype \
        --with-jpeg \
        --with-webp; \
    \
    ...; \
    \
    rm -rf /var/lib/apt/lists/*;
```

# HIERARCHIA CZYNNOŚCI PRZY BUDOWANIU OBRAZU

Usuwanie zbędnych rzeczy cz.1

ŹLE ☹️

```
ARG S6_OVERLAY_VERSION=3.1.2.1
ADD https://github.com/just-containers/s6-overlay/releases/download/v${S6_OVERLAY_VERSION}/s6-overlay-noarch.tar.xz /tmp
RUN tar -C / -Jxpf /tmp/s6-overlay-noarch.tar.xz
ADD https://github.com/just-containers/s6-overlay/releases/download/v${S6_OVERLAY_VERSION}/s6-overlay-x86_64.tar.xz /tmp
RUN tar -C / -Jxpf /tmp/s6-overlay-x86_64.tar.xz
```

DOBRE 😊

```
ARG S6_OVERLAY_VERSION=3.1.2.1
RUN set -x && curl -sL -o /tmp/s6-overlay-noarch.tar.xz https://github.com/just-containers/s6-overlay/releases/download/v${S6_OVERLAY_VERSION}/s6-overlay-noarch.tar.xz \
    && curl -sL -o /tmp/s6-overlay-x86_64.tar.xz https://github.com/just-containers/s6-overlay/releases/download/v${S6_OVERLAY_VERSION}/s6-overlay-x86_64.tar.xz \
    && tar -C / -Jxpf /tmp/s6-overlay-noarch.tar.xz \
    && tar -C / -Jxpf /tmp/s6-overlay-x86_64.tar.xz \
    && rm -rf /tmp/*.xz
```

# HIERARCHIA CZYNNOŚCI PRZY BUDOWANIU OBRAZU

Usuwanie zbędnych rzeczy cz.2

**ŹLE** 😞

```
RUN apt-get update && apt-get install -y vim
```

**DOBRCZE** 😊

```
RUN apt-get update && apt-get install -y --no-install-recommends vim=2:8.2.3995-1ubuntu2.1 \  
|&& apt-get clean \  
|&& rm -rf /var/lib/apt/lists/*
```



# HIERARCHIA CZYNNOŚCI PRZY BUDOWANIU OBRAZU

Usuwanie zbędnych rzeczy cz.3 -  
hardcore

```
# persistent dependencies
RUN set -eux; \
    apt-get update; \
    apt-get install -y --no-install-recommends \
# Ghostscript is required for rendering PDF previews
    ghostscript \
    ; \
    rm -rf /var/lib/apt/lists/*

# install the PHP extensions we need (https://make.wordpress.org/
RUN set -ex; \
    \
    savedAptMark="$(apt-mark showmanual)"; \
    \
    apt-get update; \
    apt-get install -y --no-install-recommends \
        libfreetype6-dev \
        libicu-dev \
        libjpeg-dev \
        libmagickwand-dev \
        libpng-dev \
        libwebp-dev \
        libzip-dev \
    ; \
    \
    docker-php-ext-configure gd \
```

```
    [ "$extDir" ]; \
# set apt-mark's "manual" list so that "purge --auto-remove" will remove all build dependencies
    apt-mark auto '.*' > /dev/null; \
    apt-mark manual $savedAptMark; \
    ldd "$extDir"/*.so \
        | awk '/=>/ { print $3 }' \
        | sort -u \
        | xargs -r dpkg-query -S \
        | cut -d: -f1 \
        | sort -u \
        | xargs -rt apt-mark manual; \
    \
    apt-get purge -y --auto-remove -o APT::AutoRemove::RecommendsImportant=false; \
    rm -rf /var/lib/apt/lists/*; \
    \
```

Ref: <https://github.com/docker-library/wordpress/blob/master/latest/php8.2/fpm/Dockerfile>

# HIERARCHIA CZYNNOŚCI PRZY BUDOWANIU OBRAZU

## Podsumowanie

- Zaczynamy od komend, które mają najmniejsze prawdopodobieństwo zmiany
- Najpierw kopiujemy i pobieramy zależności w Dockerfile, a potem kopiujemy kod aplikacji
- Zmiana argumentów i zmiennych także wpływa na proces budowania obrazu
- Staramy się minimalizować ilość warstw
- Staramy się używać jak najmniejszego obrazu produkcyjnego



# .DOCKERIGNORE

## Co to właściwie „Build context”?

- Polecenia **docker build** lub **docker buildx build** tworzą obrazy Dockera z pliku Dockerfile i „kontekstu”
- Kontekst budowania to zbiór plików znajdujących się w PATH lub URL określony jako argument pozycyjny polecenia budowania:

```
$ docker build .  
...  
#16 [internal] load build context  
#16 sha256:23ca2f94460dcba5b3c3edbaaa933281a4e0ea3d92fe295193e4df44dc68f85  
#16 transferring context: 13.16MB 2.2s done  
...
```

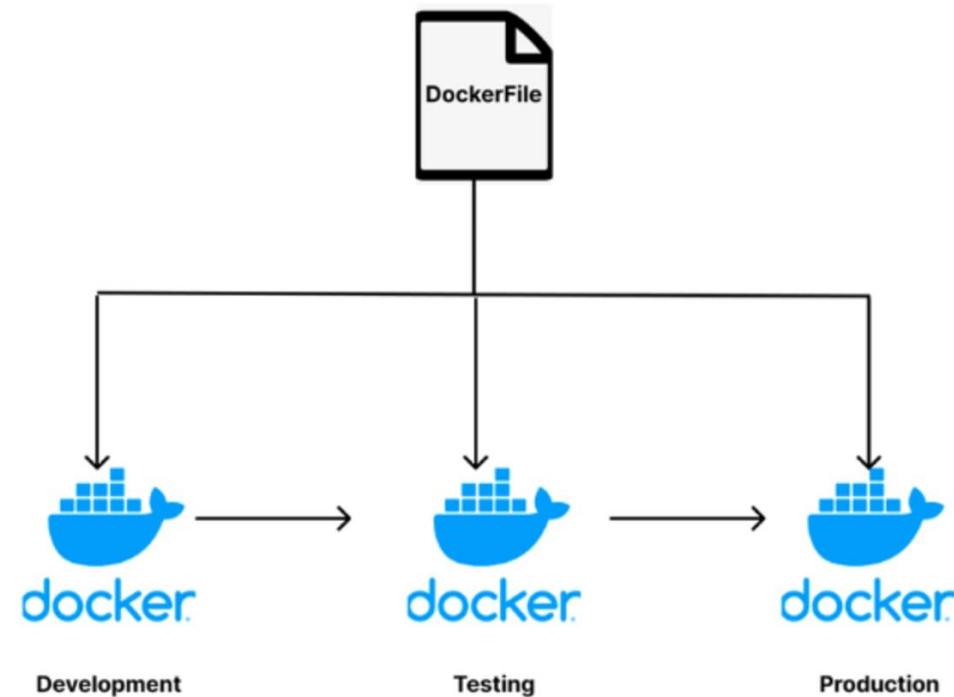
# .DOCKERIGNORE

## Podsumowując:

- Powinien być zbliżony do pliku .gitignore
- Zawiera elementy, które nie powinny znaleźć się w dockerze (np. folder *.git*)
- Zawiera zbudowane artefakty (np. *pliki JS*)
- Zawiera zależności (np. *vendor, node\_modules*)
- Zawiera pliki z sekretami
- Zawiera foldery z testami, raportami, i dokumentacją

# MULTI-STAGE

- Używanie kilku obrazów do budowania aplikacji
- Pozwala w łatwy sposób zoptymalizować docelowy rozmiar obrazu
- Wynik jednej kompilacji może być przeniesiony do innego obrazu w obrębie pliku Dockerfile
- Możliwość wygenerowania kilku obrazów dockerowych
- Oszczędność miejsca poprzez reużywanie warstw w cache
- Bezpieczeństwo danych





**Docker  
in Cloud**



**Docker  
In My  
Machine**



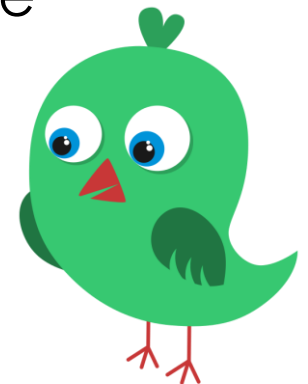
DEMO6




# OZNACZANIE OBRAZÓW – TAGOWANIE I LABELLING

# ~~TAG LATEST W PROJEKTACH~~

- Domyślny tag, gdy nie podamy żadnego
- Nie wskazuje na ostatni wypchany/najnowszy obraz dockera
- Używany głównie w lokalnym testowaniu i procesach CI/CD
- Lepiej używać unikalnego tagu np. *timestamp/sformatowana data* lub *commit\_id*
- Dodatkowo warto mieć 2 repozytoria – jedno deweloperskie, drugie produkcyjne





**ubuntu** DOCKER OFFICIAL IMAGE · 1B+ · 10K+

Ubuntu is a Debian-based Linux operating system based on free software.


[Overview](#)[Tags](#)

### Quick reference

- Maintained by:  
Canonical and Tianon (Debian Developer)
- Where to get help:  
the Docker Community Slack, Server Fault, Unix & Linux, or Stack Overflow

### Supported tags and respective Dockerfile links

- 18.04, bionic-20221215, bionic
- 20.04, focal-20221130, focal
- 22.04, jammy-20221130, jammy, latest
- 22.10, kinetic-20221130, kinetic, rolling
- 23.04, lunar-20221216, lunar, devel
- 14.04, trusty-20191217, trusty



**groovy** DOCKER OFFICIAL IMAGE · 50M+ · 137


Apache Groovy is a multi-faceted language for the Java platform.

[Overview](#)[Tags](#)

Sort by Newest

Filter Tags

TAG

[latest](#) 

Last pushed 16 days ago by [doijanky](#)


DIGEST

[c26e8835d72a](#)  
[b5d5ca7e0e72](#)  
[76c29a3e8e5f](#)  
+2 more...

OS/ARCH

linux/amd64  
linux/arm/v7  
linux/arm64/v8

TAG

[jdk17-alpine](#) 

Last pushed 3 days ago by [doijanky](#)

DIGEST

[70d532c7045c](#)

OS/ARCH

linux/amd64



# TEN SAM OBRAZ DLA DEWELOPERÓW I PRODUKCJI?

- Przydatny przy projektach z bardzo dużą ilością zależności pozaprojektowych
- Łatwy w wykonaniu dzięki budowaniu multi-stage
- Pozwala na uniknięcie duplikacji kodu i pomyłek
- Budowanie obrazu dla deweloperów i produkcji w kilku wersjach na podstawie jednego Dockerfile



```
ARG PHP_FPM_VERSION
FROM php:${PHP_FPM_VERSION} as main-php
# Set system ENV
LABEL org.label-schema.url
LABEL org.label-schema.version
LABEL org.label-schema.schema-version
LABEL org.label-schema.name
LABEL php-fpm-version=${PHP_FPM_VERSION}
ARG VCS_URL="--repository-url--"
LABEL org.label-schema.vcs-url="${VCS_URL}"
ENV BASEPATH /var/www/public
ENV LANG C.UTF-8
ENV LC_ALL C.UTF-8
SHELL ["/bin/bash", "-o", "pipefail", "-c"]
```

```
RUN bash -c "\
if [[ $PHP_VERSION =~ \"7.2\" ]]; then \
    curl -sL -o /usr/local/bin/cachetool https://gordalina.github.io/cachetool/downloads/cachetool-${CACHETOOL_5_VERSION}.phar; \
elif [[ $PHP_VERSION =~ \"7.0\" ]]; then\
    curl -sL -o /usr/local/bin/cachetool https://gordalina.github.io/cachetool/downloads/cachetool-${CACHETOOL_3_VERSION}.phar; \
elif [[ $PHP_VERSION =~ \"7.1\" ]]; then\
    curl -sL -o /usr/local/bin/cachetool https://gordalina.github.io/cachetool/downloads/cachetool-${CACHETOOL_4_VERSION}.phar; \
elif [[ $PHP_VERSION =~ \"7.3\" ]]; then\
    curl -sL -o /usr/local/bin/cachetool https://gordalina.github.io/cachetool/downloads/cachetool-${CACHETOOL_7_VERSION}.phar; \
elif [[ $PHP_VERSION =~ \"7.4\" ]]; then\
    curl -sL -o /usr/local/bin/cachetool https://gordalina.github.io/cachetool/downloads/cachetool-${CACHETOOL_7_VERSION}.phar; \
else \
    curl -sL -o /usr/local/bin/cachetool https://gordalina.github.io/cachetool/downloads/cachetool-${CACHETOOL_8_VERSION}.phar; \
fi; \
chmod a+x /usr/local/bin/cachetool; \
cachetool -v" \
&& curl -sL -o /usr/local/bin/php-fpm-healthcheck https://raw.githubusercontent.com/renatomefi/php-fpm-healthcheck/${PHP_FPM_HEA
&& chmod +x /usr/local/bin/php-fpm-healthcheck
```

```
68 # First, try installing the old way, else switch to PHP 7.4 style
69 ( \
70     docker-php-ext-configure gd \
71         --with-freetype-dir=/usr \
72         --with-jpeg-dir=/usr \
73         --with-zlib-dir=/usr \
74         --with-png-dir=/usr \
75         --with-webp-dir=/usr \
76     || \
77     docker-php-ext-configure gd \
78         --with-freetype \
79         --with-jpeg \
80         --with-webp \
81 ); \
```

```
ARG PHP_FPM_VERSION
```

```
FROM php:${PHP_FPM_VERSION} as main-php
```

```
RUN set -ex; \
```

```
| | | \
```

```
| | | apt-get update && apt-get install -y \
```

```
| | | cron \
```

```
....
```

```
FROM main-php as dev-image
```

```
RUN bash -c 'echo -e "pass=admin123" > /etc/modules_config.conf'
```

```
FROM main-php as prod-image
```

```
RUN bash -c 'echo -e "pass=SuperTajneNieDoZlamanaiHaslo" > /etc/modules_config.conf'
```

```
COPY sensitive_configuration /etc/module/module.conf
```

# ADNOTACJE OBRAZÓW OCI (DAWNIEJ LABEL SCHEMA )

- **Open Container Initiative** - tworzy standard dla obrazów i kontenerów
- Pozwala na uruchamianie tego samego obrazu przy pomocy różnych implementacji kontenerów
- <https://github.com/opencontainers/image-spec/blob/main/annotations.md>
- <http://label-schema.org/rc1/> - deprecated
- Schemat nazewnictwa etykiet (*labels*) dla obrazów dockerowych
- Zawierają m.in. czas kompilacji, wersje, autorów, dokumentacje obrazu, licencje, dostawcy, źródło (np. *git*)



:latest

:1-stable

:1.41.1

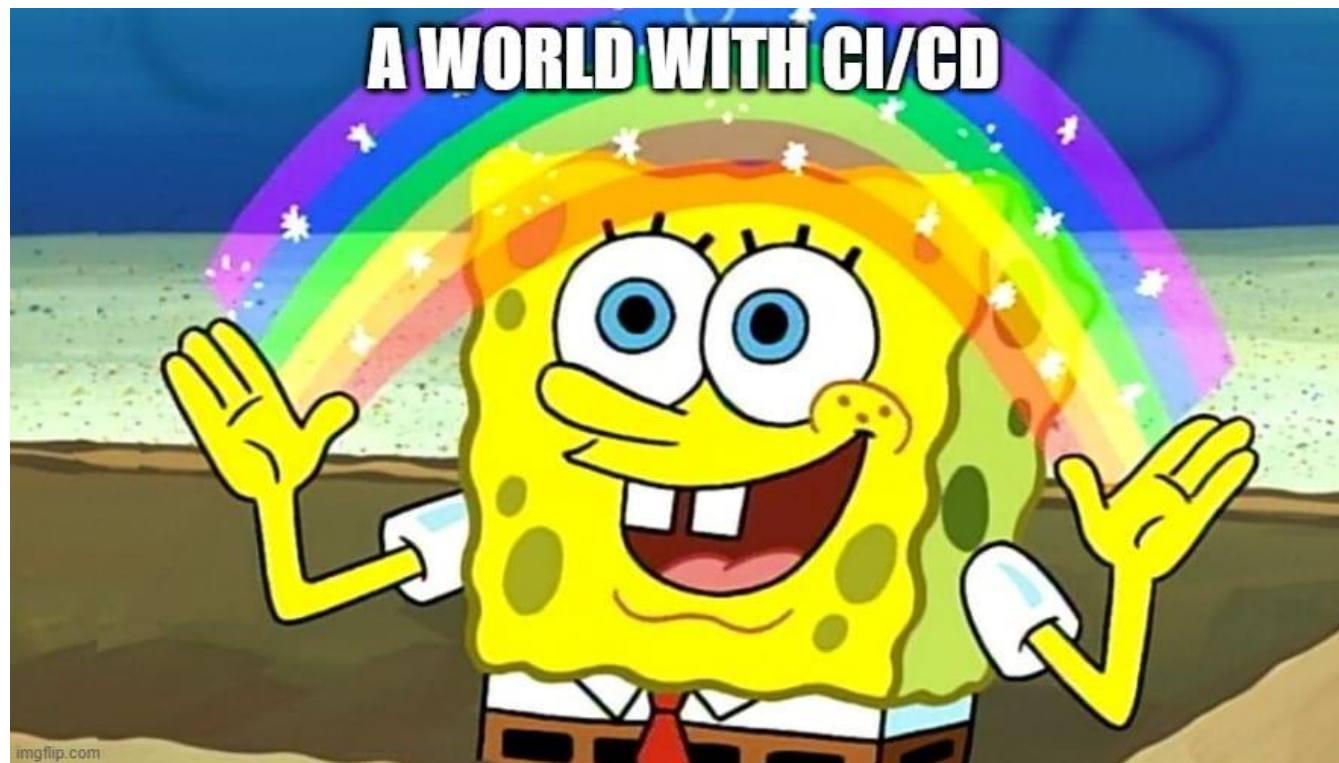
@sha256:82becede498899e

ngflip.com



# DEMO7





NASZ PRZYJACIEL – SYSTEM CI/CD

# SZABLONY POTOKÓW CI/CD

- Jeden centralny punkt zawierający szablony budowania
- Parametryzacja szablonów budowania
- Kompatybilność wsteczna
- Na pewno w twoim CI/CD to jest – przykład GitLab  
<https://docs.gitlab.com/ee/ci/yaml/index.html#extends>
- ... a jeżeli nie, to są jeszcze pluginy – przykład Jenkins  
<https://www.jenkins.io/doc/book/pipeline/shared-libraries/>

# JAK TO MOŻE WYGLĄDAĆ

## Definicja szablonu (repo1)

```
.docker-build-template2:
  image: docker:stable
  variables:
    GIT_DEPTH: 1
    destination_image: destination-image
    ~~~~~
  before_script:
    - create buildx and docker login
  script:
    - |
      echo "" >> ${dockerfile}
      echo "ARG BUILD_DATE
      ARG VCF_REF
      ARG BUILD_VERSION
      LABEL org.label-schema.version=\"${BUILD_VERSION}\" \\
      org.label-schema.build-date=\"${BUILD_DATE}\" \\
      org.label-schema.vcs-ref=\"${VCF_REF}\"" >> ${dockerfile}
    - |
      docker buildx build \
        -t ${docker_registry}/${destination_image}:${destination_specific_tag} \
        --build-arg VCS_URL=${CI_PROJECT_URL} \
        --build-arg VCF_REF=${CI_COMMIT_SHORT_SHA} \
        --build-arg BUILD_DATE=$(date -u +'%Y-%m-%dT%H:%M:%SZ') \
        --build-arg BUILD_VERSION=${CI_COMMIT_REF_NAME} \
        --build-arg BUILD_TAG=${CI_COMMIT_TAG} \
        ${custom_docker_build_param} \
        -f ${dockerfile}
```

## Budowanie obrazu (repo2)

```
include:
  - project: 'repo1'
    ref: master
    file: '/template-jobs.yml'

build-image:
  extends: .docker-build-template2
  stage: build
  variables:
    destination_image: super-image
```



# BUDOWANIE DOCKERA BEZ DOCKERA

- Kiedy nie możemy mieć dostępu bezpośrednio do socketu dockera
- Pakiety <https://github.com/guinetools/img> lub <https://github.com/GoogleContainerTools/kaniko> to umożliwiają
- Raczej dla nieskomplikowanych obrazów
- Metoda używana głównie w klastrach Kubernetes





# WHEN YOUR BOSS TELLS YOU



# DEMO 8



# Q&A

A w międzyczasie prosba o wypełnienie ankiety poszkoleniowej:

<https://bit.ly/3GILMVo>

A jak chcecie sprawdzić, czy to nie wirus to dodajcie + na końcu 😊

<https://bit.ly/3GILMVo+>