

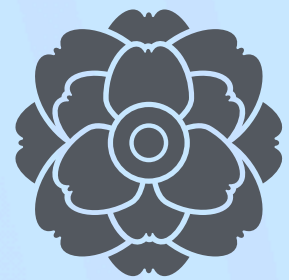
Programowanie Aplikacji w Chmurze Obliczeniowej

Laboratorium nr 7

Konfiguracja i wykorzystanie builder-ów buildx. Budowa obrazów dla wielu architektur sprzętowych. Zarządzanie danymi cache z procesu budowania.

Dr inż. Sławomir Przyłucki
s.przylucki@pollub.pl





Podstawy działania buildx - cz. I

Sterowniki Buildx to konfiguracje określające, jak i gdzie działa backend BuildKit. Ustawienia sterownika są konfigurowalne i umożliwiają precyzyjną kontrolę danego buildera. Buildx obsługuje następujące sterowniki:

CLI

```
> docker info | grep buildx
buildx: Docker Buildx (Docker Inc.)
Path: /Users/slawek/.docker/cli-plugins/docker-buildx
```

```
> docker buildx ls
```

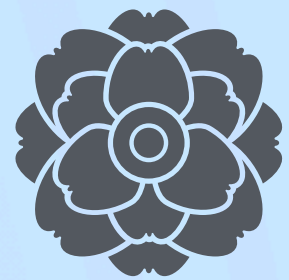
NAME/NODE	DRIVER/ENDPOINT	STATUS	BUILDKIT	PLATFORMS
default	docker			
_ default	_ default	running	v0.16.0	linux/arm64, linux/amd64, linux/amd64/v2, linux/riscv64, linux/ppc64le, linux/s390x, linux/386, linux/mips64le, linux/mips64
desktop-linux*	docker			
_ desktop-linux	_ desktop-linux	running	v0.16.0	linux/arm64, linux/amd64, linux/amd64/v2, linux/riscv64, linux/ppc64le, linux/s390x, linux/386, linux/mips64le, linux/mips64

- **docker:** używa biblioteki BuildKit dołączonej do demona Docker. (**DOMYŚLNY**)
- **docker-container:** tworzy dedykowany kontener BuildKit za pomocą Dockera.
- **kubernetes:** tworzy pod-y BuildKit w klastrze Kubernetes.
- **remote:** łączy się bezpośrednio z ręcznie zarządzanym demonem BuildKit.

<https://docs.docker.com/build/drivers/>



Architektura i zasady funkcjonowania BuildKit/buildx były przedmiotem wykładów 4 oraz 5 —> podstawy teoretyczne dla zagadnień omawianych na tym laboratorium



Podstawy działania buildx - cz. I

Containers

Images

Volumes

Builds

Docker Scout

Extensions

Builds

Build history

Active builds

Search

Selected builder

desktop-linux

Import builds

Builder settings

Docker Desktop

Selected builder

The default builder used when you start a build. [Learn more](#)

desktop-linux

docker

Running

27 minutes ago

Available builders

Inspect and manage your builders. [Learn more](#)

default

docker

Running

desktop-linux

docker

Running

2 days ago

desktop-linux

Version

v0.18.2

Status

Running

Driver

docker

History

Full history supported

Platform

Multi-platform ready

Storage limit

Used

Shared

0 Bytes

333 MB

272.4 MB

Regular

Local sources

Updated less than a minute ago

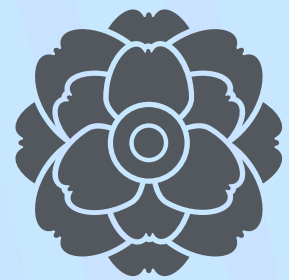
Supported platforms

arm64 amd64 amd64/v2 riscv64 ppc64le s390x 386

Advanced Information

Endpoint

desktop-linux



Tworzenie własnych builder-ów - cz. I

Składnia polecenia:

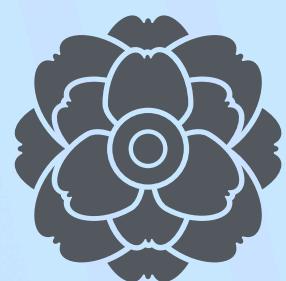
```
docker buildx create --driver docker-container --name mybuilder --use --bootstrap
```

`--driver` domyślnie używa sterownika *docker-container*. W przypadku konieczności użycia innego sterownika. Należy określić go w poleceniu.

`--name` można pominąć, w tym przypadku buildx samodzielnie wygeneruje nazwę (np.: „reverent_wilbur”)

`--use` można również pominąć, ale wtedy później należy wskazać builder (np.: `docker buildx use mybuilder`). Alternatywnie, można określać Builder jako część polecenia budowania obrazów (np.: `docker build ... --builder mybuilder`)

`--bootstrap` informuje buildx, aby natychmiast utworzył i uruchomił kontener Docker dla instancji BuildKit. W przeciwnym razie kontener jest tworzony (ang. bootstrapped) w tzw. sposób „leniwy” czyli przy pierwszym budowaniu obrazu



Tworzenie własnego builder-a, cz. II

1

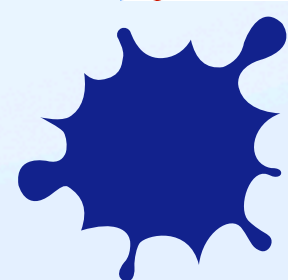
Utworzenie buildera o nazwie *testbuilder* z wykorzystaniem sterownika *docker-container*

```
~  
> docker buildx create --name testbuilder --driver docker-container --bootstrap  
[+] Building 1.3s (1/1) FINISHED  
=> [internal] booting buildkit  
=> => pulling image moby/buildkit:buildx-stable-1  
=> => creating container buildx_buildkit_testbuilder0  
testbuilder
```

Zero na końcu nazwy oznacza pierwszy węzeł w danej instancji builder-a

```
~  
> docker ps --filter name=buildx_buildkit_testbuilder0
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
99603c9356f3	moby/buildkit:buildx-stable-1	"buildkitd --allow-i..."	3 minutes ago	Up 3 minutes		buildx_buildkit_testbuilder0

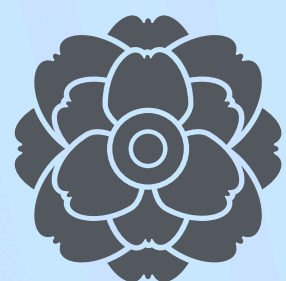


W poleceniu powyżej został utworzony i uruchomiony builder *lab7builder* ale **NIE JEST** on ustawiony jako domyślny Builder (nie dodano argumentu `--use`)



Określony builder można konfigurować poprzez dodawanie/usuwanie kolejnych węzłów skojarzonych z wybranym sterownikiem oraz deklarowanie, jaka architektura wspierana przez węzeł ma być wykorzystywana podczas budowania obrazu wieloplatformowego - przykład na wykładzie nr 5.

<https://docs.docker.com/build/building/multi-platform/#multiple-native-nodes>



Tworzenie własnego builder-a - cz. III

2

Sprawdzenie poprawności konfiguracji utworzonego builder-a

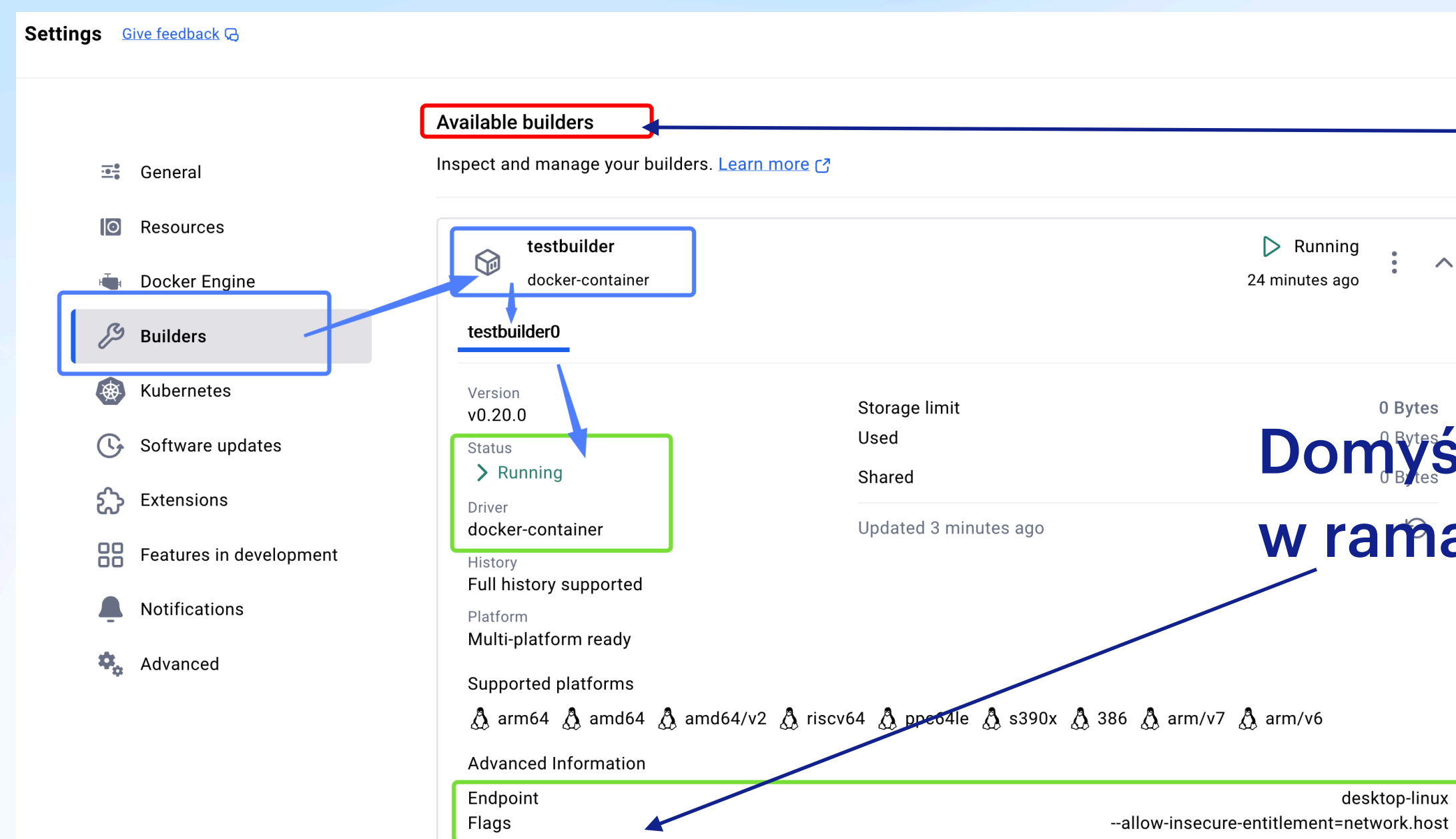
CLI

```
~  
> docker buildx ls
```

NAME/NODE	DRIVER/ENDPOINT	STATUS	BUILDKIT	PLATFORMS
testbuilder	docker-container			
_ testbuilder0	_ desktop-linux	running	v0.20.0	linux/amd64 (+2), linux/arm64, linux/arm (+2), linux/ppc64le, (3 more)
default	docker			
_ default	_ default	running	v0.18.2	linux/amd64 (+2), linux/arm64, linux/ppc64le, linux/s390x, (2 more)
desktop-linux*	docker			
_ desktop-linux	_ desktop-linux	running	v0.18.2	linux/amd64 (+2), linux/arm64, linux/ppc64le, linux/s390x, (2 more)



Węzły wchodzące w skład danego builder-a można definiować tak w trakcie tworzenie jak i późniejszych poleceń. Służą do tego opcje: `--node`, `--append`, `--leave`

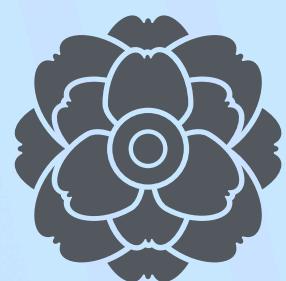


Builder jest dostępny ale nie jest ustawiony jako domyślny

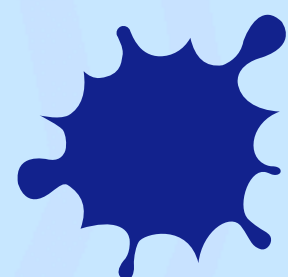
Domyślna konfiguracja węzła w ramach danego builder-a

GUI

<https://docs.docker.com/reference/cli/docker/buildx/create/>



Budowanie obrazu w oparciu o uruchomiony builder - cz. I

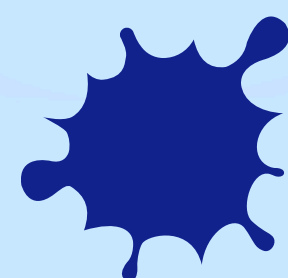
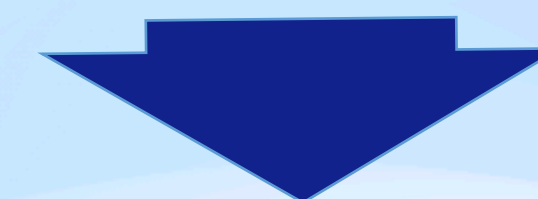


W przykładzie wykorzystywane są pliki dostępne jako plik **lab7_examples.zip**

1

Ustawienie (ew. sprawdzenie) czy builder *testbuilder* jest ustawiony jako domyślny

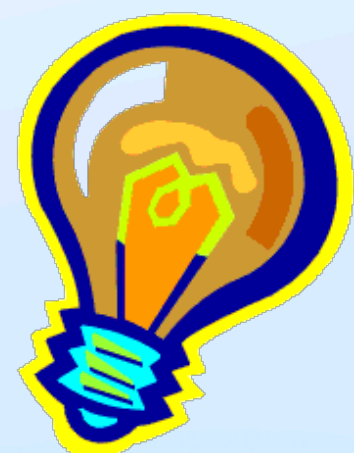
```
~/examples/L7  
> docker buildx use testbuilder
```



Przed realizacją proszę **ZALOGOWAĆ SIĘ** w środowisku Docker (polecenie `docker login ...`)

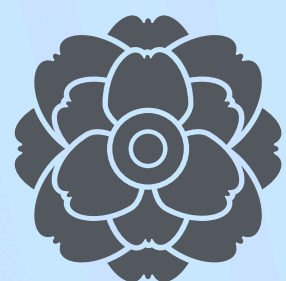
```
~/examples/L7  
> docker buildx ls
```

NAME/NODE	DRIVER/ENDPOINT	STATUS	BUILDKIT	PLATFORMS
testbuilder*	docker-container			
_ testbuilder0	_ desktop-linux	running	v0.20.0	linux/amd64 (+2), linux/arm64, linux/arm (+2), linux/ppc64le, (3 more)
default	docker			
_ default	_ default	running	v0.18.2	linux/amd64 (+2), linux/arm64, linux/ppc64le, linux/s390x, (2 more)
desktop-linux	docker			
_ desktop-linux	_ desktop-linux	running	v0.18.2	linux/amd64 (+2), linux/arm64, linux/ppc64le, linux/s390x, (2 more)



Deklarując (uruchamiając) proces budowania obrazu w oparciu o buildx za pomocą CLI, można użyć opcjonalnego flagi `--builder` lub zmiennej środowiskowej `BUILDKIT_BUILDER`, aby określić, który Builder ma być użyty. Jeśli nie określono builder-a w ten sposób, użyty będzie domyślny Builder (ustawiony poleceniem `docker buildx use ...`)

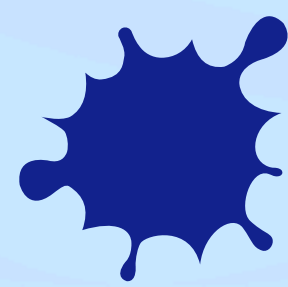
<https://docs.docker.com/build/builders/#selected-builder>



Budowanie obrazu w oparciu o uruchomiony builder - cz. II

2

Należy zbudować obrazu, który pozwoli na uruchomienie kontenera na dwóch architekturach sprzętowych: amd64 oraz arm64 oraz przesłać go do swojego repozytorium obrazów na DockerHub



!!!! proszę zmienić nazwę obrazu na zgodny ze swoim kontem na DockerHub !!!!

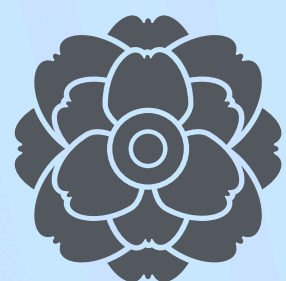
```
~/examples/L7  
> docker buildx build -q -f Dockerfile_multi -t docker.io/spg51/lab:archtest --platform linux/amd64,linux/arm64 --push .  
sha256:639601e3531936bebff20fdffe7dcd3ef7f949e7979b9bafc8b0e1d19ba846f5
```

skrót dla `--output=type=registry`, automatycznie przesyła rezultat budowy (w przykładzie, obraz kontenera) do zewnętrznego rejestru obrazów zgodnie z nazwą obrazu.



Jak sprawdzić, za pomocą polecenia CLI, że obraz został zbudowany i przesłany do właściwego rejestru obrazu oraz że jest on przestosoany dla wybranych dwóch architektur

??????



Budowanie obrazu w oparciu o uruchomiony builder - cz. III

3

Wraz z buildx dostępne jest polecenie (B.CZĘSTO WYKORZYSTYWANE) do sprawdzenia „grubego” manifestu dla danego obrazu: `docker buildx imagetools inspect`

<https://docs.docker.com/reference/cli/docker/buildx/imagetools/inspect/>

~/examples/L7

```
> docker buildx imagetools inspect docker.io/spg51/lab:archtest
```

Name: docker.io/spg51/lab:archtest

MediaType: application/vnd.oci.image.index.v1+json

Digest: sha256:639601e3531936bebff20fdffe7dcd3ef7f949e7979b9bafc8b0e1d19ba846f5

Manifests:

Name: docker.io/spg51/lab:archtest@sha256:87c2ce9a6ddcfe129f472f6adee6d5d72f102654c5469d76f2ff92c88bcff36a

MediaType: application/vnd.oci.image.manifest.v1+json

Platform: linux/amd64

Name: docker.io/spg51/lab:archtest@sha256:ce5033b02c6375d0a901691e6b73cfd210c4a7061f42e1642f9b368d1136c73c

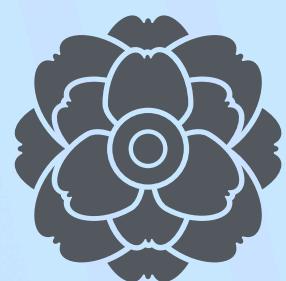
MediaType: application/vnd.oci.image.manifest.v1+json

Platform: linux/arm64

CLI



Pobierając obraz wieloarchitekturowy na lokalny komputer, pobierane są tylko warstwy zgodnie z manifestem dla architektury zgodnej z architekturą tego komputera. Jeśli taki obraz zostanie przesłany do innego repozytorium to przesłanie zostanie obraz tylko dla tej jednej architektury. Aby móc przenosić obrazy wieloarchitekturowe pomiędzy repozytoriami można używać polecenia `docker buildx imagetools create ...`



Budowanie obrazu w oparciu o uruchomiony builder - cz. III

The screenshot shows the DockerHub interface for the repository `spg51/lab`. The `Tags` tab is selected, showing a list of tags. The `archtest` tag is highlighted, and its details are shown below, including the digest and OS/ARCH. The `docker pull` command is displayed for the `archtest` tag.

DockerHub

spg51 / [Repositories](#) / [lab](#) / [Tags](#)

Using 0 of 1 private repositories.

spg51/lab

Last pushed 23 minutes ago • Repository size: 5.1 GB

[Add a description](#)

[Add a category](#)

General **Tags** Image Management BETA Builds Collaborators Webhooks Settings

☐ Sort by Newest

TAG

[archtest](#)

Last pushed 23 minutes by [spg51](#)

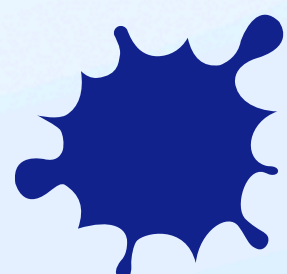
<input type="checkbox"/> Digest	OS/ARCH	Last pull	Compressed size
87c2ce9a6ddc	linux/amd64	less than 1 day	57.2 MB
ce5033b02c63	linux/arm64/v8	less than 1 day	57.13 MB

`docker pull spg51/lab:archtest`

Docker commands

To push a new tag to this repository:

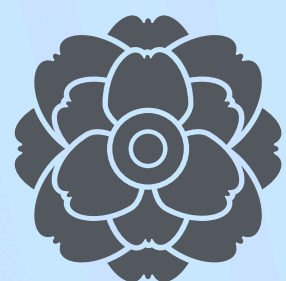
```
docker push spg51/lab:tagname
```



W najnowszych wersjach środowisku Docker dostępne jest też polecenie (wersja eksperymentalna) pozwalające na operacje na manifestach obrazów zgodnych z OCI: `docker manifest ...`

3+

Przy użyciu polecenia `docker manifest inspect <nazwa zbudowanego obrazu>` należy sprawdzić czy zbudowany obrazu wspiera dwie architektury: amd64 oraz arm64



Wykorzystanie eksporterów w Buildkit/buildx - cz. I

Eksporterzy zapisują wyniki procesu budowania obrazu (kompilacji) do określonego typu wyjściowego. Definiowanie eksportera umożliwia opcja `--output` w poleceniu CLI.

<https://docs.docker.com/build/exporters/>

Buildx obsługuje następujących eksporterów:

**Najczęściej
wykorzystywane
eksportery**

image: eksportuje wynik procesu budowania (kompilacji) do obrazu kontenera.

registry: eksportuje wynik kompilacji do obrazu kontenera i wysyła go do określonego rejestru.

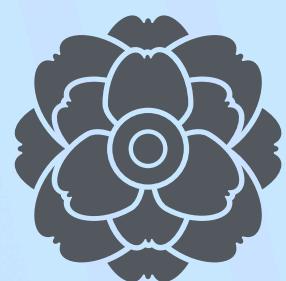
local: eksportuje główny system plików kompilacji do katalogu lokalnego.

tar: pakuje główny system plików kompilacji w lokalny plik typu tarball.

oci: eksportuje wynik kompilacji do lokalnego systemu plików w formacie układu obrazu OCI.

docker: eksportuje wynik kompilacji do lokalnego systemu plików w formacie Docker Image Specification v1.2.0.

cacheonly: nie eksportuje danych wyjściowych kompilacji, ale uruchamia kompilację i tworzy zestaw danych typu cache (pamięć podręczną).



Wykorzystanie eksporterów w Buildkit/buildx - cz. II

Aby określić eksporter, używana jest następująca składnia poleceń:

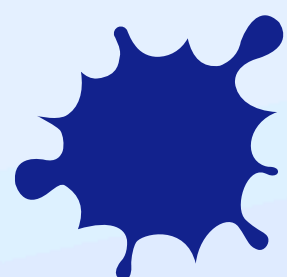
```
docker buildx build --tag <registry>/<image> --output type=<TYPE> .
```



Obrazy budowania przy użyciu sterownika *docker* są **automatycznie** ładowane do **lokalnego magazynu obrazów**.

"Klasyczny" lokalny magazyn obrazów silnika Docker nie obsługuje obrazów wieloplatformowych. Konieczne jest przekonfigurowanie środowiska Docker na magazyn obrazów containerd.

<https://docs.docker.com/build/building/multi-platform/#prerequisites>



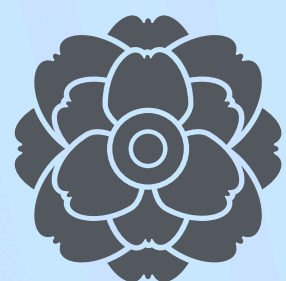
Buildx CLI automatycznie używa eksportera *docker* i ładuje obraz do lokalnego magazynu obrazów, również jeśli użyte są opcje: `--tag` oraz `--load`

lub

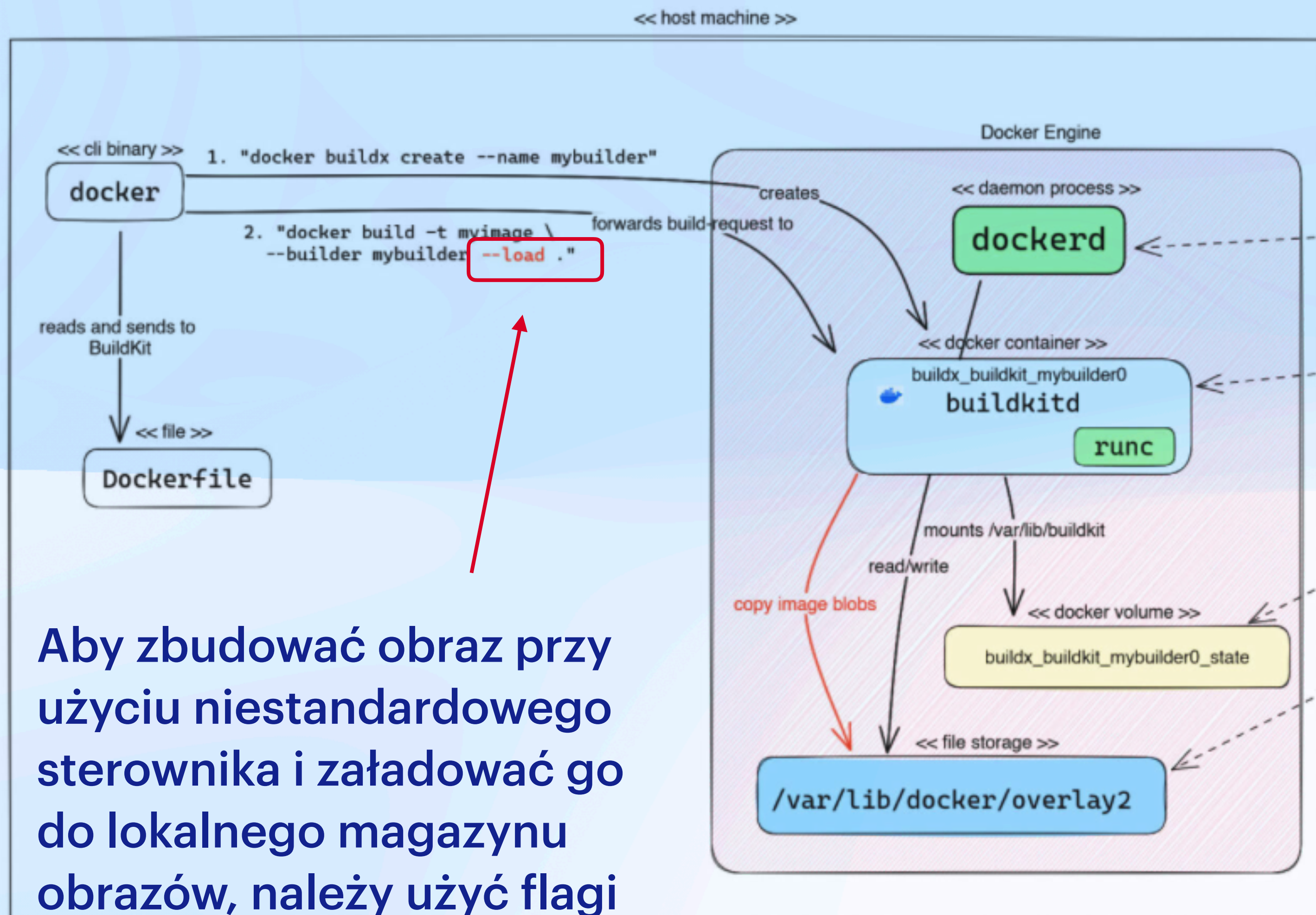
```
docker buildx build --output type=docker,name=<registry>/<image> .
```

```
docker buildx build --tag <registry>/<image> --load .
```

<https://docs.docker.com/build/exporters/#load-to-image-store>



Wykorzystanie eksporterów w Buildkit/buildx - cz. III



Aby zbudować obraz przy użyciu niestandardowego sterownika i załadować go do lokalnego magazynu obrazów, należy użyć flagi

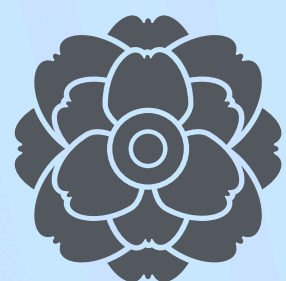
`--load` wraz z poleceniem `docker build buildx`

1 Zarządzanie lokalne - cache obrazów oraz oraz uruchamianie kontenerów na ich podstawie

2 Buduje obrazy i jest zarządzany przez buildx CLI

3 Zawiera blob-y elementów pamięci podręcznej BuildKit| utworzonych przez np. `buildx_buildkit_mybuilder0`

4 Zawiera blob-y warstw obrazu

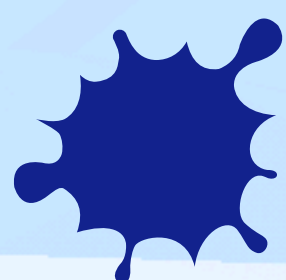


Wykorzystanie eksporterów w Buildkit/buildx - cz. II

Aby przesłać zbudowany obraz do zdalnego rejestru kontenerów, należy użyć eksportera *registry* lub *image*. Wykorzystanie opcji `--push` w poleceniach interfejsu CLI buildx powoduje, że BuildKit przekaże zbudowany obraz do określonego rejestru

```
docker buildx build --tag <registry>/<image> --push .
```

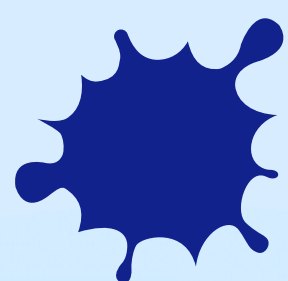
<https://docs.docker.com/build/exporters/#push-to-registry>



Z oczywistych powodów, wykorzystanie niestandardowego builder-a, np. `docker-container` umożliwia budowanie obrazów wieloplatformowych bez przełączania się na magazyn oparty o containerd.

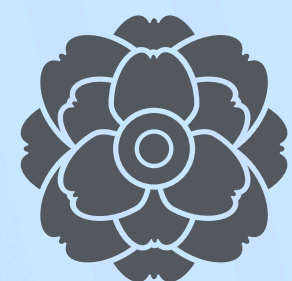


Wykorzystując niestandardowy sterownik NIE MA SENSU przysyłać obrazów do lokalnego magazynu obrazów



Istnieje możliwość jednoczesnego korzystania z więcej niż jednego eksportera

<https://docs.docker.com/build/exporters/#multiple-exporters>



Dane cache w Buildkit/buildx - cz. I

BuildKit automatycznie buforuje wynik kompilacji we własnej wewnętrznej pamięci podręcznej. Dodatkowo, BuildKit obsługuje również eksport pamięci podręcznej kompilacji do zewnętrznej lokalizacji, umożliwiając import danych cache w przyszłych kompilacjach (procesach budowania obrazów).

<https://docs.docker.com/build/cache/backends/>

Buildx obsługuje następujące backend-y pamięci podręcznej

inline: dane cache procesu budowania obrazu są umieszczane w obrazie. Wymaga określenia eksportera obrazów w poleceniach `docker buildx build`

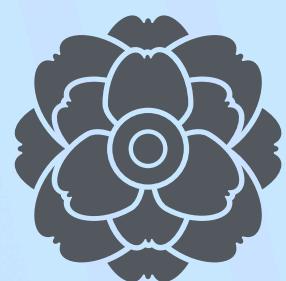
registry: dane cache umieszczane są w osobnym obrazie i wysyłane do dedykowanej lokalizacji oddzielonej od miejsca docelowego dla obrazu (innego niż deklarowane, główne wyjście polecenia `docker buildx build`)

local: zapisuje dane cache do lokalnego katalogu w systemie plików.

gha: przesyła dane cache procesu budowania obrazu do pamięci podręcznej systemu GitHub (beta).

s3: przesyła dane cache do zasobnika AWS S3 (alfa).

azblob: przesyła dane cache do Azure Blob Storage (alfa).



Dane cache w Buildkit/buildx - cz. II

Zewnętrzna pamięć podręczna jest **NIEZBĘDNA** w środowiskach budowania CI/CD. Takie środowiska zazwyczaj mają niewielką lub żadną trwałość między uruchomieniami, ale nadal ważne jest, aby czas budowy obrazów dzięki danym cache był jak najkrótszy.

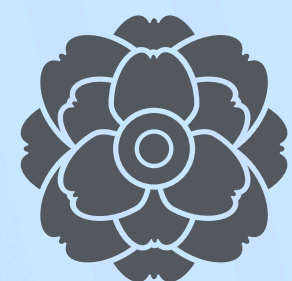


Domyślny sterownik buildx *docker* obsługuje backend-y `inline`, `local`, `registry` oraz `gha`, ale **TYLKO WTEDY gdy używany jest lokalny magazyn obrazów oparty na containerd**. Inne backend-y cache wymagają wybrania innego sterownika.

Przykład wykorzystania danych cache (oddzielny obraz z danymi cache przesyłany do zewnętrznego registry):

```
docker buildx build --push -t <registry>/<image> \  
--cache-to type=registry,ref=<registry>/<cache-image>[,parameters...] \  
--cache-from type=registry,ref=<registry>/<cache-image>[,parameters...] .
```

<https://docs.docker.com/build/cache/backends/#command-syntax>



Dane cache w Buildkit/buildx - cz. III

Podczas generowania wyjścia dla pamięci podręcznej, argument `--cache-to` może być uzupełniona przez opcję trybu do definiowania, które warstwy należy uwzględnić w wyeksportowanej pamięci podręcznej. W buildx (w Buildkitd) dostępne są dwa tryby:

- tryb **min** - jest to tryb **domyślny**, tylko warstwy eksportowane do wynikowego obrazu są buforowane,
- tryb **max** - wszystkie warstwy są eksportowane

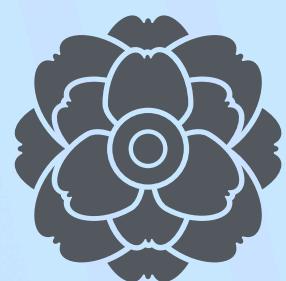


W trybie minimalnej pamięci podręcznej (domyślnym) buforowane są tylko warstwy, które są eksportowane do wynikowego obrazu, podczas gdy w trybie maksymalnej pamięci podręcznej buforowane są wszystkie warstwy, nawet te z kroków pośrednich.



Wymienione wyżej tryby są dostępne dla wszystkich backend-ów z wyjątkiem inline. Ten backend wspiera wyłącznie tryb min.

<https://docs.docker.com/build/cache/backends/#cache-mode>

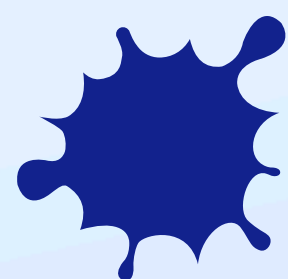


Dane cache w Buildkit/buildx - przykład - cz. I

1

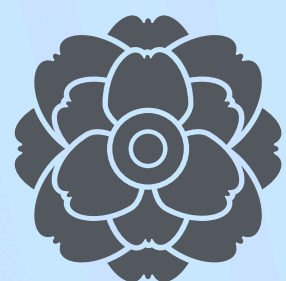
Przykład bazuje na przykładzie wykorzystywanym wcześniej (Dockerfile_multi). Obraz będzie budowany w oparciu o sterownik *docker-container*, utworzony wcześniej oraz skonfigurowany jako domyślny builder. Dane cache będą eksportowane do DockerHub (backend registry) z wykorzystaniem trybu max (cache obejmuje wszystkie etapy budowania)

```
~/examples/L7  
> docker buildx build -f Dockerfile_multi --platform linux/amd64,linux/arm64 -t docker.io/spg51/lab:cachetest --push \\\n--cache-to type=registry,ref=docker.io/spg51/cachedata1,mode=max \\\n--cache-from type=registry,ref=docker.io/spg51/cachedata1 .
```



Uwaga: przy pierwszej budowie obrazu pojawi się błąd. Wynika on z oczywistej przyczyny, prze pierwszej budowie nie ma możliwości pobrania danych cache (cache-from)

Na kolejnych slajdach pokazany jest wynik działania polecenia podczas drugiego (kolejnego) procesu budowania obrazu



Dane cache w Buildkit/buildx - przykład cz. III

2A

```
~/examples/L7
> docker buildx build -f Dockerfile_multi --platform linux/amd64,linux/arm64 -t docker.io/spg51/lab:cachetest --push \
--cache-to type=registry,ref=docker.io/spg51/cachedata1,mode=max \
--cache-from type=registry,ref=docker.io/spg51/cachedata1 .
[+] Building 55.2s (28/28) FINISHED
=> [internal] load build definition from Dockerfile_multi
=> => transferring dockerfile: 377B
=> [linux/amd64 internal] load metadata for docker.io/library/node:23.1.0-alpine3.20
=> [linux/amd64 internal] load metadata for docker.io/library/node:23.1.0
=> [linux/arm64 internal] load metadata for docker.io/library/node:23.1.0-alpine3.20
=> [linux/arm64 internal] load metadata for docker.io/library/node:23.1.0
=> [internal] load .dockerignore
=> => transferring context: 2B
=> importing cache manifest from docker.io/spg51/cachedata1
=> => inferred cache manifest type: application/vnd.oci.image.index.v1+json
```

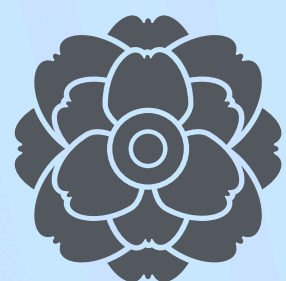
Analiza metadanych
dla obrazów bazowych

Pobieranie i wykorzystanie danych cache
z poprzedniego procesu budowania
obrazów

2B

```
=> => transferring context: 6.00kB
=> CACHED [linux/arm64 build1 2/5] RUN mkdir -p /var/node
=> CACHED [linux/arm64 build1 3/5] WORKDIR /var/node
=> CACHED [linux/arm64 build1 4/5] ADD src ./
=> CACHED [linux/arm64 build1 5/5] RUN npm install
=> CACHED [linux/arm64 prod 2/3] COPY --from=build1 /var/node /var/node
=> CACHED [linux/arm64 prod 3/3] WORKDIR /var/node
=> CACHED [linux/amd64 build1 2/5] RUN mkdir -p /var/node
=> CACHED [linux/amd64 build1 3/5] WORKDIR /var/node
=> CACHED [linux/amd64 build1 4/5] ADD src ./
=> CACHED [linux/amd64 build1 5/5] RUN npm install
=> CACHED [linux/amd64 prod 2/3] COPY --from=build1 /var/node /var/node
=> CACHED [linux/amd64 prod 3/3] WORKDIR /var/node
```

Potwierdzenie poprawności
wykorzystania pobranych
danych cache



Dane cache w Buildkit/buildx - przykład cz. IV

2C

```
=> exporting to image
=> => exporting layers
=> => exporting manifest sha256:87c2ce9a6ddcfe129f472f6adee6d5d72f102654c5469d76f2ff92c88bcff36a
=> => exporting config sha256:8a1cb08f0679ee12367e4df16e425e45c90666f82c705542211398cd9e08f81a
=> => exporting attestation manifest sha256:1607ffa2176a759d8e417b81ff943275e6f32efd33c348a44fbc932ff8e9f4d9
=> => exporting manifest sha256:ce5033b02c6375d0a901691e6b73cfd210c4a7061f42e1642f9b368d1136c73c
=> => exporting config sha256:a2770dc7387a197f2742b34b2a07aaffbdf2874b53818d865a43927758e16905
=> => exporting attestation manifest sha256:3a9a0a965949411f325a8081fb7995f783f4d6ba8ffcd57c1768c2c0e459d8d1
=> => exporting manifest list sha256:330f489affef9ead7b3db961f1aa8644f924e9b3b68d64f3b471f11752c98697
=> => pushing layers
=> => pushing manifest for docker.io/spg51/lab:cachetest@sha256:330f489affef9ead7b3db961f1aa8644f924e9b3b68d64f3b471f11752c98697
=> [auth] spg51/lab:pull,push token for registry-1.docker.io
```

Eksportowanie warstw
obrazu zgodnie ze
zdeklarowanym
eksporterem *image*

2D

```
=> exporting cache to registry
=> => preparing build cache for export
=> => writing layer sha256:13f4a8312f8b6817ec48c62e598f9ddfd8800b444e110b4ce198b718cc65d843
=> => writing layer sha256:14b7e0bb74a837e8395628a2f0dbb92e489b216072bd03fa0d951c68a1cbb60b
=> => writing layer sha256:1a3f1864ec54b1398987bbe673e93d8b09842ecd51e86ab87d64857b70d188b1
=> => writing layer sha256:2112e5e7c3ff699043b282f1ff24d3ef185c080c28846f1d7acc5ccf650bc13d
```

Eksportowanie warstw dla obiektu z danymi cache

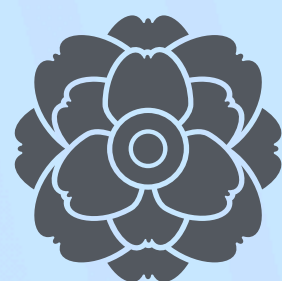
Informacje dostępne tylko gdy kontener z silnikiem
Buildkit jest uruchomiony w Docker Desktop

```
.....
=> => writing layer sha256:fdddf066a59bbdf8fceed81036df2e90447e04f51615c0055b2c5db2fa3456431
=> => writing config sha256:3bcc46aa02533ba79d5ba8c9d524096464514bc084f766a9712b3863f3a9f019
=> => writing cache manifest sha256:5f1ba052a256080f438a13ad2fe40a583a920d0b9b8346bb0b4b2bae6bc05c7e
=> [auth] spg51/cachedata1:pull,push token for registry-1.docker.io
```

```
View build details: docker-desktop://dashboard/build/testbuilder/testbuilder0/a1nrhyjixlulx2pla5nzquhtr
```



Dane cache w trybie max (szczególnie gdy stosowana jest wieloetapowa budowa obrazów mogą zajmować dużo miejsca. Proszę mieć to na uwadze gdy korzysta się ze zdalnych (np. chmurowych) rejestrów obrazów



Dane cache w Buildkit/buildx - przykład cz. V

3

DockerHub

spg51

▼

🔍


Search by repository name

All content



▼



Name	Last Pushed ↑	Contains	Visibility
spg51/cachedata1	about 1 hour ago	IMAGE	Public

spg51 / [Repositories](#) / [cachedata1](#) / [Image-management](#)

spg51/cachedata1 



Last pushed about 1 hour ago • Repository size: 901.1 MB

[Add a description](#)  

[Add a category](#)  

General Tags **Image Management BETA** Builds Collaborators Webhooks Settings

[Where to start?](#) [Report an issue](#)

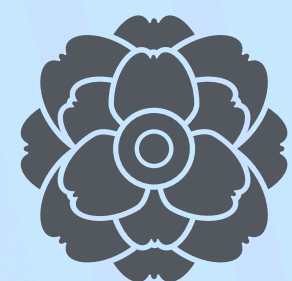
 Digest	Tags	Media type	OS/ARCH	Size	Last pushed
<input checked="" type="checkbox"/> sha256:5f1ba052a256	 latest	Image Index	-	901 MB	about 1 hour

Dane cache jako oddzielny obraz w rejestrze obrazów na DockerHub



Proszę zwrócić uwagę na wielkość obrazu cache - również stosowanie backend-u *inline* powiększa obraz !!!! - zawsze należy rozważyć rodzaj stosowanych polityki zarządzania danymi cache oraz zapoznać się z metodami garbage collection

<https://docs.docker.com/build/cache/garbage-collection/>



PAwChO — Laboratorium 7

Zadanie do wykonania - NIEOBOWIĄZKOWE - cz. I

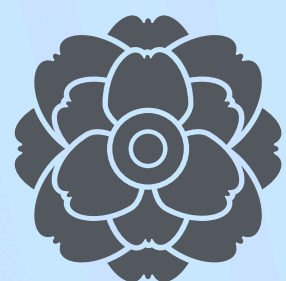
Na laboratorium 3 przedstawiona była metoda wykorzystania lokalnego rejestru obrazów w postaci kontenera uruchomionego w oparciu o obraz registry

https://hub.docker.com/_/registry

Należy wykorzystać aplikację przygotowaną jako rozwiązanie zadania obowiązkowego z laboratorium nr 5.

Zadanie polega na:

- Uruchomieniu własnego rejestru obrazów w oparciu o obraz *registry* (proszę nazwać go: *registry7*)
- Utworzeniu i skonfigurowaniu własnego builder-a wykorzystującego sterownik *docker-container*
- Na bazie tego builder-a zbudowaniu obrazu aplikacji dla architektur: amd64 oraz arm64 i przesłanie go do swojego własnego rejestru obrazów *registry7*
- W trakcie budowania ma być zadeklarowane przechowywanie danych cache utworzonych w trybie max w *registry7* jako oddzielny obraz



Zadanie do wykonania - NIEOBOWIĄZKOWE - cz. II

W sprawozdaniu z zadania należy:

- Przeprowadzić minimum dwa procesy budowania obrazów
- Za pomocą poznanych poleceń wyświetlić zawartość manifestu potwierdzającego, że obraz jest przeznaczony dla dwóch wymaganych architektur
- W informacjach o procesie budowania wskazać na miejsca potwierdzające poprawne (założone w poleceniu) zarządzanie danymi cache

Jeśli wystąpią problemy (błędy) przy realizacji któregoś z etapów zadania - proszę podać przyczynę ze wskazaniem odpowiednich informacji z dokumentacji środowiska Docker oraz przedstawić pomysł na rozwiązanie napotkanego problemu



Możliwość otrzymania dodatkowych punktów (+50%)

Ponieważ dane cache zajmują dużo miejsca to powstaje pytanie: jak usunąć obraz cache z rejestru *registry7* ????? Pytanie można też uogólnić: jak usuwać dowolny obraz z rejestru w postaci kontenera zbudowanego w oparciu o obraz reggistry ????