



POLITECHNIKA
LUBELSKA

Lublin 22.05.2022
Ryszard Rogalski

Sprawozdanie - laboratorium I Programowanie fullstack w chmurze obliczeniowej

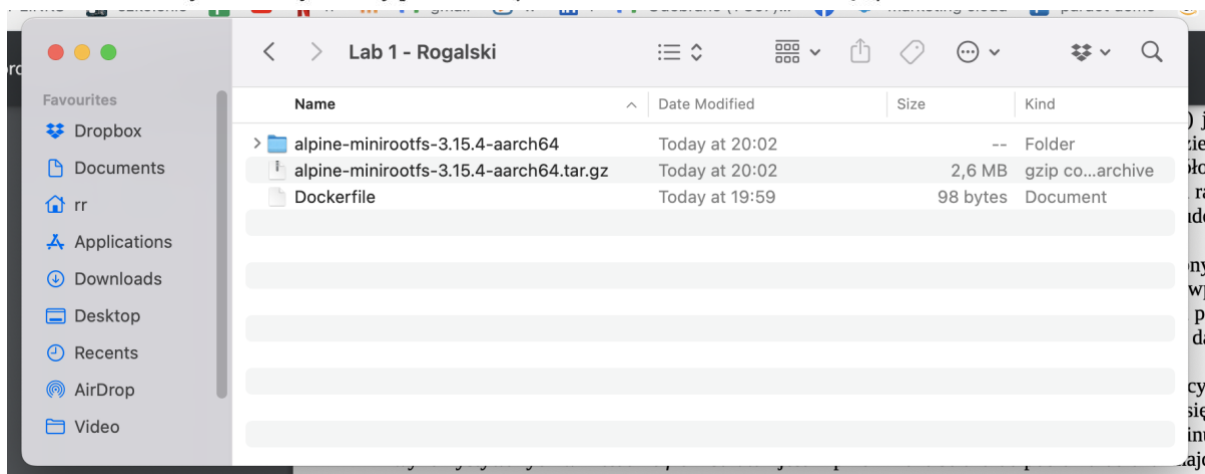
Prowadzący zajęcia

dr inż. Sławomir Przyłucki

Jako niezbędny komponent, z całą pewnością będzie potrzebny minimalny system operacyjny. Wiele dystrybucji oferuje takie minimalne obrazy w postaci plików TAR (należy odwołać się do dokumentacji konkretnej dystrybucji). Jedną z najpopularniejszych dystrybucji Linuxa, wykorzystywanych w metodzie *from scratch* jest Alpine Linux. Strona do pobrania obrazu znajduje się pod adresem: <https://alpinelinux.org/downloads/> (sekcja MINI ROOT FILESYSTEM). Mając pobrany ten plik, można utworzyć prosty Dockerfile, taki jak ten, przedstawiony na rysunku 1.1:

```
Dockerfile_scratch x
Dockerfile_scratch > ...
1 FROM scratch
2 ADD files/alpine-minirootfs-3.14.2-aarch64.tar.gz /
3 CMD ["/bin/sh"]
```

Rys. 1.1. Przykładowy plik Dockerfile do zbudowania obrazu metodą „from scratch”



```
Dockerfile X
Dockerfile > ...
1 FROM scratch
2 ADD alpine-minirootfs-3.15.4-aarch64.tar.gz /
3 CMD ["/bin/sh"]
4 RUN apk add apache2 php
```

```
Lab 1 - Rogalski --zsh-- 87x18
rr@Ryszards-MacBook-Air Lab 1 - Rogalski % docker build -t scratch/apache_php .
[+] Building 1.3s (6/6) FINISHED
=> [internal] load build definition from Dockerfile                                0.0s
=> => transferring dockerfile: 36B                                              0.0s
=> [internal] load .dockerignore                                                0.0s
=> => transferring context: 2B                                                  0.0s
=> [internal] load build context                                                0.1s
=> => transferring context: 2.62MB                                             0.1s
=> [1/2] ADD alpine-minirootfs-3.15.4-aarch64.tar.gz /                        0.1s
=> [2/2] RUN apk add apache2 php                                              1.0s
=> exporting to image                                                         0.1s
=> => exporting layers                                                         0.1s
=> => writing image sha256:f8b580da0c4b134a6250b7e6c6b0d7de4e87a5109469dec14151 0.0s
=> => naming to docker.io/scratch/apache_php                                0.0s

Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn ho
w to fix them
rr@Ryszards-MacBook-Air Lab 1 - Rogalski %
```

P1.3. W zadaniu P1.1 zbudowany został plik konfiguracyjny *Dockerfile* dla serwera HTTP_PHP. Proszę, wzorując się na teście przedstawionym na rysunku 1.6, określić czas budowania obrazu wykorzystując klasyczny silnik *build* oraz nowy silnik *BuildKit*. Proszę pamiętać o opcji wyłączającej wykorzystanie pamięci cache w procesach *build*.

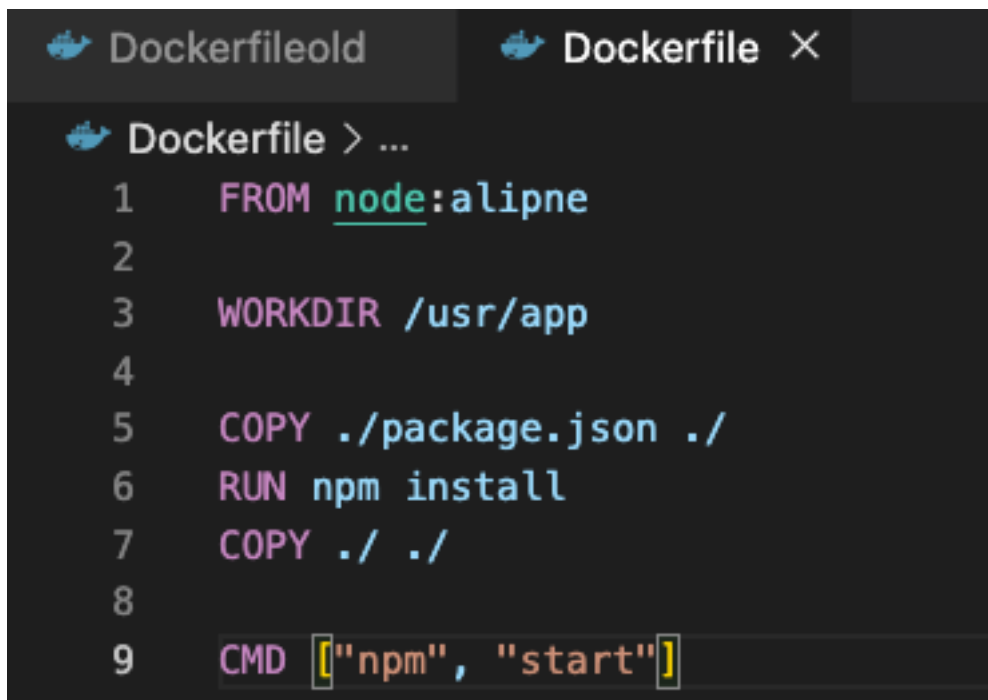
Proszę przedstawić otrzymane wyniki i odpowiedzieć na dwa poniższe pytania:

1. Czy a jeśli tak to dlaczego, otrzymane wyniki są gorsze (mniejszy zysk czasy, mniejszy efekt równoleglenia) od tych, które zostały uzyskane na rysunku 1.6?
2. Czy można i w jaki sposób można dokonać zmian, które poprawiły wyniki na korzyść nowego silnika BuildKit (wzmocniły efekt równoleglenia)?

```
rr@Ryszards-MacBook-Air Lab 1 - Rogalski % DOCKER_BUILDKIT=0
rr@Ryszards-MacBook-Air Lab 1 - Rogalski % time docker build -q --no-cache .
sha256:e210a6d20f647d1219df461877b1c3c81c4093615915b58896f0b57d2f3f9c2a
docker build -q --no-cache . 0.10s user 0.04s system 6% cpu 2.053 total
rr@Ryszards-MacBook-Air Lab 1 - Rogalski % DOCKER_BUILDKIT=1
rr@Ryszards-MacBook-Air Lab 1 - Rogalski % time docker build -q --no-cache .
sha256:abb0fe833b551be486df8e9b7b67116c62082c040ee692ca4cd38d95af51c569
docker build -q --no-cache . 0.10s user 0.04s system 9% cpu 1.519 total
rr@Ryszards-MacBook-Air Lab 1 - Rogalski %
```

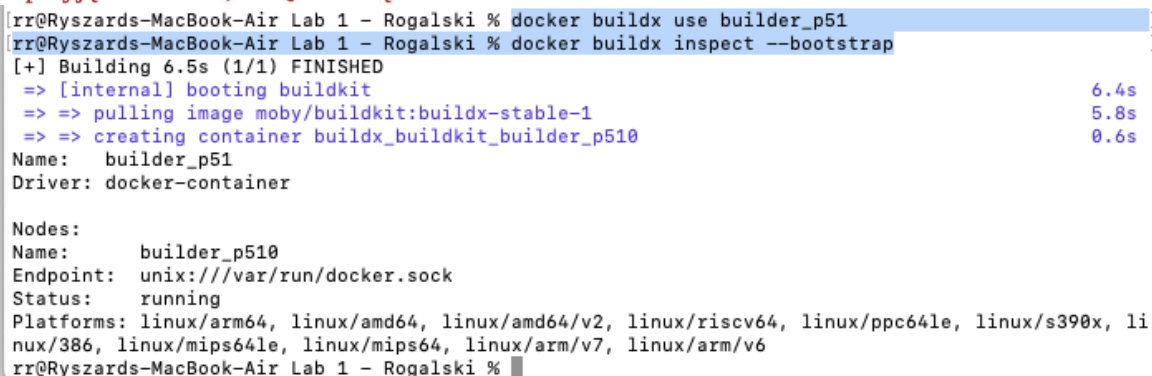
Wyniki procentowo są gorsze (mamy tutaj niewielki zysk, ok. 25%), ponieważ budujemy jeden prosty obraz, a więc teoretycznie mamy mniej miejsc do zyskania przewagi. Można spodziewać się znacznych przyspieszeń w przypadkach, w których możliwe jest równoległe przetwarzanie warstw obrazu.

P5.1. Przedstawione wyżej 5 kroków prowadzących do zbudowania obrazów dla trzech wybranych architektur sprzętowych wykorzystywały banalny plik konfiguracyjny *Dockerfile* z rysunku 1.21 (plik ten oraz wymagane kodyźródłowe są dostępne na moodle).



```
Dockerfileold Dockerfile X
Dockerfile > ...
1 FROM node:alpine
2
3 WORKDIR /usr/app
4
5 COPY ./package.json ./
6 RUN npm install
7 COPY ./ ./
8
9 CMD ["npm", "start"]
```

1. Utworzyć środowisko budowania obrazów wieloplatformowych na bazie wrapera buildx. Proszę przyjąć założenie, że QEMU będzie zainstalowane lokalnie.



```
rr@Ryszards-MacBook-Air Lab 1 - Rogalski % docker buildx use builder_p51
rr@Ryszards-MacBook-Air Lab 1 - Rogalski % docker buildx inspect --bootstrap
[+] Building 6.5s (1/1) FINISHED
=> [internal] booting buildkit 6.4s
=> => pulling image moby/buildkit:buildx-stable-1 5.8s
=> => creating container buildx_buildkit_builder_p510 0.6s
Name: builder_p51
Driver: docker-container

Nodes:
Name: builder_p510
Endpoint: unix:///var/run/docker.sock
Status: running
Platforms: linux/arm64, linux/amd64, linux/amd64/v2, linux/riscv64, linux/ppc64le, linux/s390x, li
nux/386, linux/mips64le, linux/mips64, linux/arm/v7, linux/arm/v6
rr@Ryszards-MacBook-Air Lab 1 - Rogalski %
```

2. Na podstawie Dockerfile z rysunku 1.21 proszę zbudować obrazy dla czterech wybranych architektur sprzętowych.

