POLITECHNIKA
LUBELSKA
WYDZIAŁ ELEKTROTECHNIKI
I INFORMATYKI

# Programowanie Aplikacji w Chmurze Obliczeniowej

Laboratorium 5. Związki pomiędzy instrukcjami ARG oraz ENV. Wskazówki i dobre praktyki podczas tworzenia plików Dockerfile. Automatyczna kontrola poprawności działania aplikacji, Budowa obrazów metodą „od podstaw” – Zadanie do wykonania obowiązkowe.

Marek Prokopiuk

grupa dziekańska: 6.7
numer albumu: 097710

Dr inż. Sławomir Przyłucki

Lublin rok 2024

# 1. Utworzenie pliku Dockerfile

Zadanie polegało na utworzeniu pliku Dockerfile, który wykorzystywać miał metodę wieloetapowego budowania obrazów. Należało zrealizować je w dwóch etapach. W pierwszym z nich trzeba było wykorzystać obraz bazowy „scratch", natomiast w drugim obraz bazowy Nginx w dowolnej wersji.

Wersja aplikacji miała być określona w poleceniu *docker build* poprzez nadanie wartości zmiennej VERSION definiowanej przez instrukcję ARG. W drugim etapie aplikacja z etapu pierwszego miała zostać skopiowana na serwer HTTP i ustawiona tak, aby być domyślnie uruchamiana i wyświetlana jako strona domyślna. Oprócz tego miało zostać uwzględnione sprawdzanie poprawności działania poprzez wykorzystanie HEALTHCHECK. Poniżej widać treść utworzonego pliku Dockerfile. Wprowadzone zostały komentarze informujące co się dzieje w danej linijce.

```
# ETAP 1 Budowa aplikacji Node.js
FROM scratch AS etap1
ADD alpine-minirootfs-3.19.1-x86_64.tar /

# Zadeklarowany klucz, ale nie wartość
ARG BASE_VERSION

# Deklaracja zmiennej Environment
# Jeżeli BASE_VERSION nie będzie posiadało wartości
# To wersja ta będzie wpisana defaultowo jako v1
ENV APP_VERSION=${BASE_VERSION:-v1}

# Instalacja pakietów niezbędnych do realizacji testu
RUN apk add --update nodejs npm && rm -rf /var/cache/apk/*

# Deklaracja katalogu roboczego
WORKDIR /usr/app

# Kopiowanie niezbędnych zależności
COPY ./package.json ./
# Instalacja tych zależności
RUN npm install

# Kopiowanie kodu aplikacji wewnątrz obrazu
COPY ./index.js ./

#-----------------------------------------------------------------------
```

*Rysunek 1 Plik Dockerfile - etap 1*

```
#-------------------------------------------------------------------
# ETAP 2 Konfiguracja i uruchomienie nginx
FROM nginx:alpine3.19 AS etap2

# Ponowne zadeklarowanie zmiennych
ARG BASE_VERSION
ENV APP_VERSION=${BASE_VERSION:-v1}

# Instalacja curl
RUN apk add --update curl && \
    apk add --update nodejs npm && \
    rm -rf /var/cache/apk/*

# Skopiowanie z etapu 1 do katalogu domyślnego serwera HTTP
COPY --from=etap1 /usr/app /usr/share/nginx/html/

# Skopiowanie pliku konfiguracyjnego
# nginx.conf do katalogu /etc/nginx/conf.d/
COPY nginx.conf /etc/nginx/conf.d/default.conf

# Deklaracja katalogu roboczego
WORKDIR /usr/share/nginx/html

# Informacja o porcie wewnętrznym kontenera,
# na ktorym "nasłuchuje" aplikacja
EXPOSE 8080

# Informacja czy aplikacja działa
# procedura weryfikacji działania uruchomionej aplikacji
HEALTHCHECK --interval=10s --timeout=1s \
    CMD curl -f http://localhost:8080/ || exit 1

# Domyśle polecenie przy starcie kontenera
CMD ["npm", "start", "-g", "deamon off"]
```

*Rysunek 2 Plik Dockerfile - etap 2*

W etapie pierwszym budowana jest prosta aplikacja webowa w oparciu o Nodejs. Wyświetla ona takie informacje jak imię i nazwisko, adres IP serwera, na którym aplikacja jest uruchomiona, nazwę serwera oraz wersję aplikacji. Wersja aplikacja zgodnie z poleceniem jest określona w poleceniu *docker build*, poprzez nadanie wartości odpowiedniej zmiennej co będzie pokazane potem. Na poniższym rysunku widać kod pliku *index.js*, który tworzy prosty serwer i zwraca odpowiedź zawierającą informację o tym serwerze.

```
const express = require('express');
const os = require("os");

const app = express();

app.get('/', (req, res) => {

  // Pobranie adresu IP serwera
  const ip = req.socket.localAddress;

  // Pobranie nazwy hosta
  const hostname = os.hostname();

  // Wysłanie odpowiedzi z informacjami
  res.send("Zadanie do wykonania - Laboratorium 5 - Marek Prokopiuk<br>" +
           "Adres IP serwera: " + ip + "<br>" +
           "Nazwa serwera (hostname): " + hostname + "<br>" +
           "Wersja aplikacji: " + process.env.APP_VERSION);
});

app.listen(8080, () => {
  console.log('Listening on port 8080');
});
```

*Rysunek 3 Zawartość pliku index.js*

## 2. Budowanie obrazu i uruchomienie serwera

Po zrealizowaniu dwóch etapów związanych z utworzeniem odpowiedniego pliku Dockerfile, należało zbudować na jego podstawie obraz. Następnie uruchomić serwer i potwierdzić działanie nowo powstałego kontenera oraz poprawne funkcjonowanie opracowanej aplikacji.

Aplikacja realizowała wymaganą funkcjonalność. Obraz udało się odpowiednio utworzyć, a także uruchomić kontener na podstawie tego zbudowanego obrazu. Sprawdzone zostało poprawne funkcjonowanie kontenera poprzez odczytanie statusu healthy, otrzymanego po wprowadzeniu odpowiedniego polecenia. Widok z okna przeglądarki zgodnie z oczekiwaniami pokazywał konkretne dane dotyczące serwera. Widać, że wersja aplikacja jest równa v2, zgodnie z tym co zostało podane w poleceniu *docker build*. Na poniższych zrzutach ekranu widać wszystkie użyte polecenia.

```
marek2@DESKTOP-1LRNN89:~$ cd Zad_lab5
marek2@DESKTOP-1LRNN89:~/Zad_lab5$ ls
Dockerfile_Lab5  alpine-minirootfs-3.19.1-x86_64.tar  index.js  nginx.conf  package.json
marek2@DESKTOP-1LRNN89:~/Zad_lab5$ docker build --build-arg BASE_VERSION=v2 -f Dockerfile_Lab5 -t local/marek_lab5:v0 .
[+] Building 8.0s (16/16) FINISHED                                                                  docker:default
 => [internal] load build definition from Dockerfile_Lab5                                                   0.0s
 => => transferring dockerfile: 1.81kB                                                                      0.0s
 => [internal] load metadata for docker.io/library/nginx:alpine3.19                                         1.4s
 => [internal] load .dockerignore                                                                           0.0s
 => => transferring context: 2B                                                                             0.0s
 => [internal] load build context                                                                           0.0s
 => => transferring context: 792B                                                                           0.0s
 => [etap2 1/5] FROM docker.io/library/nginx:alpine3.19@sha256:e9e0c3a702f02406384e0d7b0a1d4e7aa7            0.0s
 => => resolve docker.io/library/nginx:alpine3.19@sha256:e9e0c3a702f02406384e0d7b0a1d4e7aa7fbc3c9            0.0s
 => [etap1 1/6] ADD alpine-minirootfs-3.19.1-x86_64.tar /                                                    0.0s
 => [etap1 2/6] RUN apk add --update nodejs npm && rm -rf /var/cache/apk/*                                  2.2s
 => [etap1 3/6] WORKDIR /usr/app                                                                            0.0s
 => [etap1 4/6] COPY ./package.json ./                                                                      0.0s
 => [etap1 5/6] RUN npm install                                                                             3.9s
 => [etap1 6/6] COPY ./index.js ./                                                                          0.0s
 => CACHED [etap2 2/5] RUN apk add --update curl &&     apk add --update nodejs npm &&     rm -rf        0.0s
 => [etap2 3/5] COPY --from=etap1 /usr/app /usr/share/nginx/html/                                           0.1s
 => [etap2 4/5] COPY nginx.conf /etc/nginx/conf.d/default.conf                                              0.0s
 => [etap2 5/5] WORKDIR /usr/share/nginx/html                                                               0.0s
 => exporting to image                                                                                      0.1s
 => => exporting layers                                                                                     0.1s
 => => writing image sha256:a8408927753b13411ec885674a4a461f1f4609db06a8bc8d4e8b3fd5fe5d0f73                0.0s
 => => naming to docker.io/local/marek_lab5:v0                                                              0.0s

What's Next?
  1. Sign in to your Docker account → docker login
  2. View a summary of image vulnerabilities and recommendations → docker scout quickview
marek2@DESKTOP-1LRNN89:~/Zad_lab5$
```

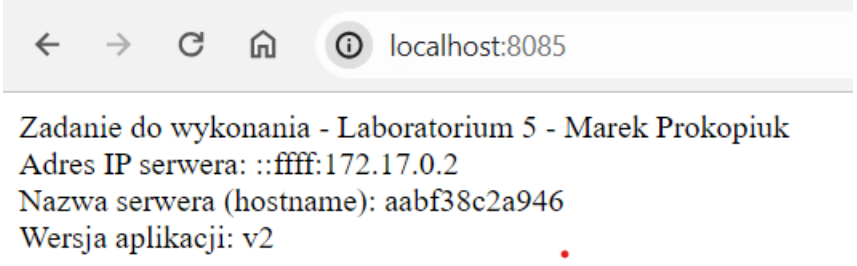
*Rysunek 4 Polecenie użyte do budowy obrazu oraz wynik jego działania*

```
marek2@DESKTOP-1LRNN89:~/Zad_lab5$ docker images
REPOSITORY         TAG       IMAGE ID       CREATED          SIZE
local/marek_lab5   v0        a8408927753b   56 seconds ago   112MB
marek2@DESKTOP-1LRNN89:~/Zad_lab5$ docker run -d -p 8085:8080 --name marek_nginxserver local/marek_lab5:v0
aabf38c2a9466e1b47fbda590c6ac0c4138d48e98f0e6ba96cdbb154096d6eb1
marek2@DESKTOP-1LRNN89:~/Zad_lab5$
```

*Rysunek 5 Polecenie tworzące kontener na podstawie obrazu. Uruchomienie serwera*

```
marek2@DESKTOP-1LRNN89:~/Zad_lab5$ docker ps --filter name=marek_nginxserver
CONTAINER ID   IMAGE              COMMAND                  CREATED         STATUS                   PORTS                                   NAMES
aabf38c2a946   local/marek_lab5:v0   "/docker-entrypoint.…"   3 minutes ago   Up 3 minutes (healthy)   80/tcp, 0.0.0.0:8085->8080/tcp   marek_nginxserver
marek2@DESKTOP-1LRNN89:~/Zad_lab5$
```

*Rysunek 6 Polecenie potwierdzające poprawne działanie kontenera*




*Rysunek 7 Potwierdzenie, że aplikacja realizuje wymaganą funkcjonalność*