POLITECHNIKA LUBELSKA

WYDZIAŁ ELEKTROTECHNIKI I INFORMATYKI

KIERUNEK STUDIÓW

INFORMATYKA

Sprawozdanie z laboratorium 5

Przedmiot: Programowanie aplikacji w chmurze obliczeniowej

Autor:

Ewelina Musińska

Gr. 6.9

Link do githuba: https://github.com/evee03/pawcho

Etap pierwszy w pliku Dockerfile (stage 1):

Dockerfile:

```
Dockerfile > ...
1   # budowanie obrazu docker
2   # tworzenie pustego obrazu scratch jako podstawy
3   FROM scratch AS builder
4
5   # dodanie minimalnego systemu alpine linux
6   ADD alpine-minirootfs-3.21.3-x86_64.tar.gz /
7
8   # instalacja zależności i aplikacji Node.js oraz npm
9   RUN apk update && \
10      apk add --no-cache nodejs npm
11
12  # definiowanie zmiennej ARG dla wersji aplikacji
13  ARG VERSION
14
15  # ustawienie zmiennej środowiskowej VERSION
16  ENV VERSION=${VERSION}
17
18  # skopiowanie pliku package.json (tak, aby npm install był wykonywany tylko wtedy, gdy zmieni się package.json)
19  COPY ./package.json /usr/app/package.json
20
21  # ustawienie katalogu roboczego
22  WORKDIR /usr/app
23
24  # instalowanie zależności opisane w package.json
25  # (jeśli zmieni się package.json, to npm install będzie wykonywany ponownie)
26  RUN npm install
27
28  # dkopiowanie aplikacji index.js
29  COPY ./index.js /usr/app/index.js
30
31  # ustawienie portu
32  EXPOSE 8080
33
34  # uruchomienie aplikacji
35  CMD ["node", "index.js"]
36
```

Rysunek 1 Dockerfile stage 1

index.js:

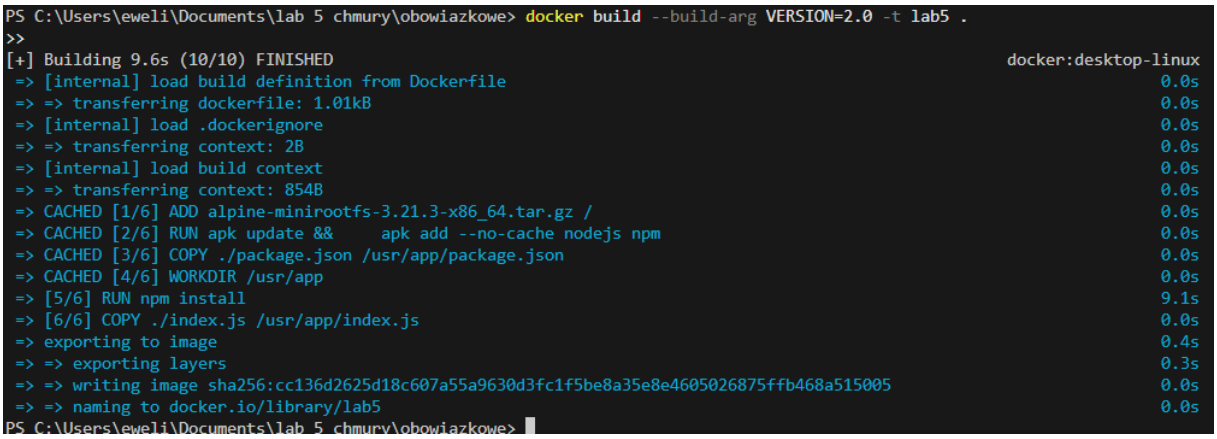
```
index.js > ...
1   // zaladowuje biblioteki express i os
2   const express = require('express');
3   const os = require('os');
4
5   // definiuje trase http get na glowna strone
6   // oraz wyswietla informacje o serwerze
7   const app = express();
8
9   app.get('/', (req, res) => {
10    const serverIP = req.connection.localAddress;
11    const serverHostname = os.hostname();
12    const appVersion = process.env.VERSION;
13
14    res.send(`
15      <h1>Lab 5 Ewelina Musińska:</h1>
16      <p>Server IP: ${serverIP}</p>
17      <p>Hostname: ${serverHostname}</p>
18      <p>Version: ${appVersion}</p>
19    `);
20  });
21
22  // uruchamia serwer na porcie 8080
23  // oraz wyswietla komunikat o uruchomieniu serwera
24  app.listen(8080, () => {
25    console.log('Listening on port 8080');
26  });
```

Rysunek 2 index.js stage 1

package.json

Rysunek 3 package.json stage 1

W tym Dockerfile tworzony jest obraz, zaczynając od pustego obrazu scratch, a następnie dodawany jest minimalny system Alpine Linux. Instalowane są zależności, takie jak Node.js i npm. Zmienna VERSION jest przekazywana jako argument i ustawiana jako zmienna środowiskowa, co pozwala na dynamiczne ustalanie wersji aplikacji. Plik package.json jest kopiowany przed plikiem aplikacji (index.js), ponieważ zmiany w tym pliku powodują, że Docker ponownie uruchomi npm install, aby zainstalować zależności. Dzięki temu, jeśli plik index.js zmienia się, ale package.json pozostaje niezmieniony, instalacja zależności nie jest wykonywana ponownie, co przyspiesza budowanie obrazu. Na końcu plik index.js jest kopiowany, a aplikacja uruchamiana na porcie 8080. Dzięki tej kolejności plików Docker efektywnie wykorzystuje mechanizm cache, optymalizując proces budowy obrazu.



Rysunek 4 Budowanie obrazu stage 1

Zbudowałam kontener, a następnie lekko zmodyfikowałam zawartość pliku index.js. Dzięki zastosowaniu mechanizmu cache w Dockerze, najpierw zostały wykonane kroki COPY package.json oraz RUN npm install, co pozwoliło na wykorzystanie wcześniej zainstalowanych zależności. Dopiero później, po zmianie pliku index.js, Docker ponownie wykonał odpowiednie kroki, uwzględniając zmiany w aplikacji.

```
PS C:\Users\eweli\Documents\lab 5 chmury\obowiazkowe> docker run -p 8080:8080 lab5
>>
Listening on port 8080
```

Rysunek 5 Uruchomienie kontenera na porcie 8080

Rysunek 6 Widok http://localhost:8080

Etap drugi w pliku Dockerfile (stage 2):

Dockerfile:

```
34  # wykorzystanie obrazu nginx
35  FROM nginx:alpine
36
37  # musze zainstalowac node.js w finalnym obrazie poniewaz potrzebny jest  do uruchomienia javascriptu
38  RUN apk add --no-cache nodejs npm
39
40  # ponownie przekazanie ARG do finalnego obrazu ponieważ
41  # zmienna ARG jest dostępna tylko w czasie budowy obrazu
42  ARG VERSION
43  ENV VERSION=${VERSION}
44
45  # kopiuje aplikację node.js zbudowana w pierwszym etapie
46  # (z katalogu roboczego /usr/app w obrazie builder)
47  COPY --from=builder /usr/app /usr/app
48
49  # kopiuje plik konfiguracyjny nginx.conf do katalogu konfiguracyjnego nginx
50  # musze ustawć nginx jako reverse proxy dla aplikacji node.js
51  # (port 8080 w kontenerze nginx jest mapowany na port 80 w kontenerze node.js)
52  COPY nginx.conf /etc/nginx/nginx.conf
53
54  # uruchamiam obie uslugi (Node.js w tle a Nginx na pierwszym planie)
55  CMD (cd /usr/app && node index.js &) && nginx -g "daemon off;"
56
57  # ustawiam port 80 dla nginx
58  EXPOSE 80
59
60  # sprawdzenie poprawności działania aplikacji
61  # (sprawdza, czy aplikacja Node.js działa na porcie 8080)
62  HEALTHCHECK --interval=30s --timeout=10s --retries=3 \
63    CMD curl -f http://localhost/ || exit 1
```

Rysunek 7 Dockerfile stage 2

Domyślnie obraz Nginx jest skonfigurowany do obsługi statycznych plików HTML z katalogu /usr/share/nginx/html. Oczekuje, że w tym katalogu znajdzie plik index.html, który będzie serwowany jako strona główna (startowa). Dlatego aby umożliwić działanie

aplikacji Node.js jako backendu i przekierowywanie żądań HTTP z Nginx do aplikacji Node.js, musiałam zmodyfikować domyślną konfiguracje Nginx.

nginx.conf

```
nginx.conf
1   # events - ile jednoczednie połączeń może być obsługiwanych przez Nginx
2   # musialam to dodać bo bez tego nie działało
3   events {
4       worker_connections 1024;
5   }
6
7   #http i server ustawia serwer http ktory będzie nasłuchiwał na porcie 80
8   # i przekazywał żądania do lokalnego serwera na porcie 8080
9   http {
10      server {
11          listen 80;
12
13          location / {
14              proxy_pass http://localhost:8080; # wszykie żądania z portu 80 będą przekazywane do lokalnego serwera na porcie 8080
15
16          }
17      }
18  }
```

Rysunek 8 nginx.conf

Pliki index.js oraz package.json pozostają bez zmian.

Buduję obraz oraz definiuję wersję:

```
PS C:\Users\eweli\Documents\lab 5 chmury\obowiazkowe> docker build --build-arg VERSION=2.0.0 -t lab5 .
>>
[+] Building 1.1s (16/16) FINISHED                                                                  docker:desktop-linux
 => [internal] load build definition from Dockerfile                                                                0.0s
 => => transferring dockerfile: 2.10kB                                                                              0.0s
 => [internal] load metadata for docker.io/library/nginx:alpine                                                     0.9s
 => [auth] library/nginx:pull token for registry-1.docker.io                                                        0.0s
 => [internal] load .dockerignore                                                                                   0.0s
 => => transferring context: 2B                                                                                     0.0s
 => [stage-1 1/4] FROM docker.io/library/nginx:alpine@sha256:4ff102c5d78d254a6f0da062b3cf39eaf07f01eec0             0.0s
 => [internal] load build context                                                                                   0.0s
 => => transferring context: 152B                                                                                   0.0s
 => CACHED [stage-1 2/4] RUN apk add --no-cache nodejs npm                                                          0.0s
 => CACHED [builder 1/6] ADD alpine-minirootfs-3.21.3-x86_64.tar.gz /                                               0.0s
 => CACHED [builder 2/6] RUN apk update &&     apk add --no-cache nodejs npm                                        0.0s
 => CACHED [builder 3/6] COPY ./package.json /usr/app/package.json                                                  0.0s
 => CACHED [builder 4/6] WORKDIR /usr/app                                                                           0.0s
 => CACHED [builder 5/6] RUN npm install                                                                            0.0s
 => CACHED [builder 6/6] COPY ./index.js /usr/app/index.js                                                          0.0s
 => CACHED [stage-1 3/4] COPY --from=builder /usr/app /usr/app                                                      0.0s
 => CACHED [stage-1 4/4] COPY nginx.conf /etc/nginx/nginx.conf                                                      0.0s
 => exporting to image                                                                                              0.0s
 => => exporting layers                                                                                             0.0s
 => => writing image sha256:08460bc53a312768357d353457b10de7068a56e371ef8d4fb027b439d0b2b948                        0.0s
 => => naming to docker.io/library/lab5                                                                             0.0s
```

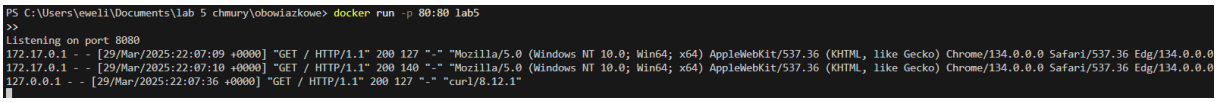
Rysunek 9 Budowanie obrazu

Uruchamiam kontener na porcie 80. W odpowiedzi dostaje dodatkowo że port 8080 również działa.

```
PS C:\Users\eweli\Documents\lab 5 chmury\obowiazkowe> docker run -p 80:80 lab5
>>
Listening on port 8080
```

Rysunek 10 Uruchamianie kontenera

Logi z serwera nginx pokazują z którego ip przyszło żądanie, rodzaj żądania http, kod statusu 200 który oznacza że wszystko poszło dobrze oraz liczbe bajtów w odpowiedzi którą serwer wysłał do klienta.



Rysunek 11 Logi nginx

Widok:

Rysunek 12 Widok http://localhost:80