POLITECHNIKA LUBELSKA
WYDZIAŁ ELEKTROTECHNIKI I INFORMATYKI

KIERUNEK STUDIÓW
INFORMATYKA

Przedmiot: Programowanie aplikacji w chmurze obliczeniowej

Sprawozdanie - Laboratorium 5

Autor:
Wiktoria Matacz
Gr.6.9

Etap pierwszy w pliku DockerLab5:

DockerLab5:

Lab5Zad > DockerLab5

```
#Tworzenie obrazu scratch jako obraz bazowy
FROM scratch AS build-stage

ADD alpine-minirootfs-3.21.3-aarch64.tar /

# Dodajemy wymagane pakiety
RUN apk add --no-cache nodejs npm

# Ustawiamy katalog roboczy
WORKDIR /app

# Kopiujemy kod aplikacji
COPY package.json package.json /app/
RUN npm install

COPY server.js /app/server.js

# Definiujemy zmienną ARG dla wersji aplikacji
ARG VERSION="unknown"

# Ustawiamy zmienną środowiskową dla wersji aplikacji
ENV APP_VERSION=$VERSION

# Uruchamiamy aplikację
CMD ["node", "server.js"]
```

Rys. 1 DockerLab5

server.js

Lab5Zad > JS server.js > ...

```
1   const express = require('express');
2   const os = require('os');
3
4   const app = express();
5   const PORT = 8080;
6
7   app.get('/', (req, res) => {
8       const ipAddress = Object.values(os.networkInterfaces())
9           .flat()
10          .filter((iface) => iface.family === 'IPv4' && !iface.internal)
11          .map((iface) => iface.address)[0] || 'Unknown';
12
13      const hostname = os.hostname();
14      const version = process.env.APP_VERSION || 'unknown';
15
16      res.send(`
17          <h1>Server Information - Lab5</h1>
18          <p><strong>IP Address:</strong> ${ipAddress}</p>
19          <p><strong>Hostname:</strong> ${hostname}</p>
20          <p><strong>Application Version:</strong> ${version}</p>
21      `);
22  });
23
24  app.listen(PORT, () => {
25      console.log(`Server running on http://localhost:${PORT}`);
26  });
27
```

Rys. 2 server.js

package.json

Lab5Zad > {} package.json > ...

```
1   {
2       "version": "1.0.0",
3       "main": "server.js",
4       "dependencies": {
5           "express": "*"
6       }
7   }
8
```

Rys. 3 package.json

W DockerLab5 został utworzony kontenerowy obraz aplikacji Node.js na bazie obrazu scratch. Scratch jest wykorzystywany jako bazowy obraz systemu plików, do którego dodano Alpine Linux. Zainstalowano wymagane pliki Node.js oraz npm, package.json i server.js zostały skopiowane. Ustawiono zmienną środowiskową APP_VERSION, która będzie przechowywać wersję aplikacji podaną jako argument `VERSION` podczas budowania obrazu. Na końcu zdefiniowano polecenie uruchamiające aplikację po starcie kontenera – CMD [“node”, “server.js”]. W pliku server.js zostały zaimportowane wymagane moduły. Następnie utworzono serwer nasłuchujący na porcie `8080`. Po otrzymaniu żądania GET na główny endpoint `/`, aplikacja zwraca stronę HTML zawierającą:

- Adres IP serwera,
- Nazwę hosta,
- Wersję aplikacji pobraną z `process.env.APP_VERSION`.

Plik `package.json` zawiera podstawowe informacje o aplikacji oraz jej zależności. W tym przypadku jedyną wymaganą biblioteką jest **Express.js**, której wersja nie została określona (`"express": "*"`, co oznacza, że zostanie zainstalowana najnowsza dostępna wersja).

```
C:\Users\Dell\Desktop\Semestr6\ProgAplChmuraObl\Lab5Zad>docker build --build-arg VERSION=1.0.0 -f DockerLab5 -t lab5app .
[+] Building 1.3s (10/10) FINISHED                                                                          docker:desktop-linux
 => [internal] load build definition from DockerLab5                                                                    0.0s
 => => transferring dockerfile: 616B                                                                                    0.0s
 => [internal] load .dockerignore                                                                                       0.0s
 => => transferring context: 2B                                                                                         0.0s
 => [internal] load build context                                                                                       0.0s
 => => transferring context: 120B                                                                                       0.0s
 => CACHED [1/6] ADD alpine-minirootfs-3.21.3-aarch64.tar /                                                             0.0s
 => CACHED [2/6] RUN apk add --no-cache nodejs npm                                                                      0.0s
 => CACHED [3/6] WORKDIR /app                                                                                           0.0s
 => CACHED [4/6] COPY package.json package.json /app/                                                                   0.0s
 => CACHED [5/6] RUN npm install                                                                                        0.0s
 => CACHED [6/6] COPY server.js /app/server.js                                                                          0.0s
 => exporting to image                                                                                                  1.1s
 => => exporting layers                                                                                                 0.7s
 => => exporting manifest sha256:0b8daed4ae58d444dbbf0e6cc5a287c9076720724O6b4e16cce78abec20990b6                        0.0s
 => => exporting config sha256:18fef8a58e792bbacde694c1ff96d62219eb588af7aa9a523a525fd4b8238918                         0.0s
 => => exporting attestation manifest sha256:6ea5e2c20d95b2e17bc922251d4c9970b233a407b4971d900b2d68a28b6d5693           0.0s
 => => exporting manifest list sha256:1574199e358b1b6820557a4ed48a0dc35d27abf0657af782ed61fa3c8c8f415a                  0.0s
 => => naming to docker.io/library/lab5app:latest                                                                       0.0s
 => => unpacking to docker.io/library/lab5app:latest                                                                    0.3s

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/1ggyuuqkvkgz0chebj7na4k0k
```

Rys. 4 Budowa obrazu

```
C:\Users\Dell\Desktop\Semestr6\ProgAplChmuraObl\Lab5Zad>docker run -p 8080:8080 lab5app
Server running on http://localhost:8080
```

Rys. 5 Uruchomienie kontenera za pomocą portu 8080

Rys.6 Przeglądarka http://localhost:8080

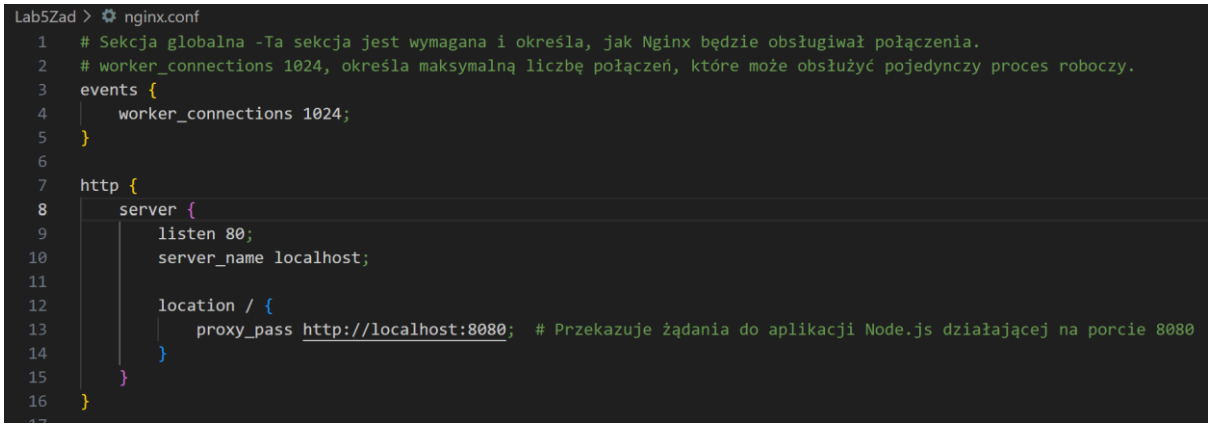
Etap drugi w pliku DockerLab5:

DockerLab5:

Lab5Zad > DockerLab5

```
27
28  # Etap 2: Nginx jako serwer HTTP
29  FROM nginx:alpine
30
31  # dodajemy pakiety
32  RUN apk add --no-cache nodejs npm
33
34  # Definiujemy zmienną ARG dla wersji aplikacji
35  ARG VERSION="unknown"
36  # Ustawiamy zmienną środowiskową dla wersji aplikacji
37  ENV APP_VERSION=$VERSION
38
39  # Kopiujemy aplikację z etapu 1 (build-stage)
40  COPY --from=build-stage /app /app
41
42  # Kopiujemy plik konfiguracji Nginx, aby ustawić stronę startową
43  COPY nginx.conf /etc/nginx/nginx.conf
44
45  # Uruchamiamy oba procesy w jednym kontenerze
46  CMD (cd /app && node server.js &) && nginx -g "daemon off;"
47
48  EXPOSE 80
49
50  # Definiujemy HEALTHCHECK - sprawdzanie stanu aplikacji
51  HEALTHCHECK --interval=30s --timeout=10s --retries=3 \
52    CMD curl --fail http://localhost:8080/ || exit 1
53
```

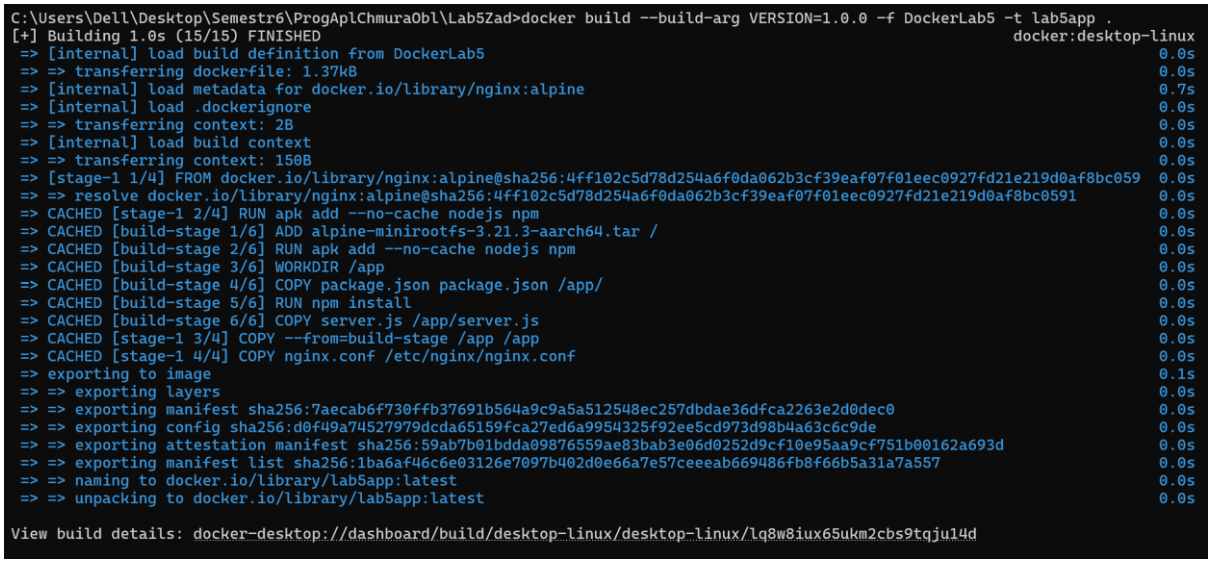
Rys. 7 DockerLab5



Rys. 8 nginx.conf

Pozostałe pliki są nie zmienione.

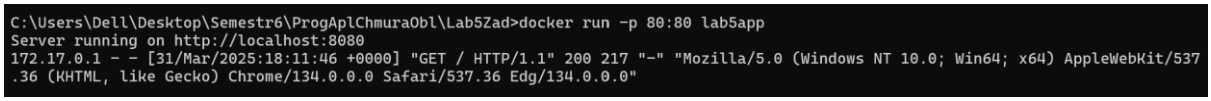
Stworzono kontenerową aplikację **Node.js**, obsługiwaną przez serwer **Nginx**, działającą w środowisku **Alpine Linux**. W drugim etapie wykorzystano obraz **nginx:alpine**, aby skonfigurować Nginx jako **serwer proxy** dla aplikacji. Dodano pakiety **Node.js** i **npm**, a następnie skopiowano do obrazu pliki aplikacji z wcześniejszego etapu. Aby Nginx mógł poprawnie obsługiwać ruch HTTP, skopiowano również plik `nginx.conf`, w którym skonfigurowano serwer. W pliku konfiguracyjnym określono, że serwer będzie działał na porcie `80`, a wszystkie żądania kierowane na ten port będą przekazywane do aplikacji **Node.js**, która działa na porcie `8080`. Dzięki temu użytkownicy mogą łączyć się z aplikacją bezpośrednio przez port `80`, zamiast odwoływać się do domyślnego portu aplikacji Node.js. Aby zapewnić prawidłowe działanie obu procesów w jednym kontenerze, w sekcji `CMD` uruchomiono aplikację w tle (`node server.js &`), a następnie uruchomiono serwer Nginx. Dodatkowo dodano mechanizm **HEALTHCHECK**, który co 30 sekund sprawdza, czy aplikacja działa poprawnie. Jeśli serwer **Node.js** nie odpowiada, system może wykonać odpowiednie działania, np. zrestartować kontener.



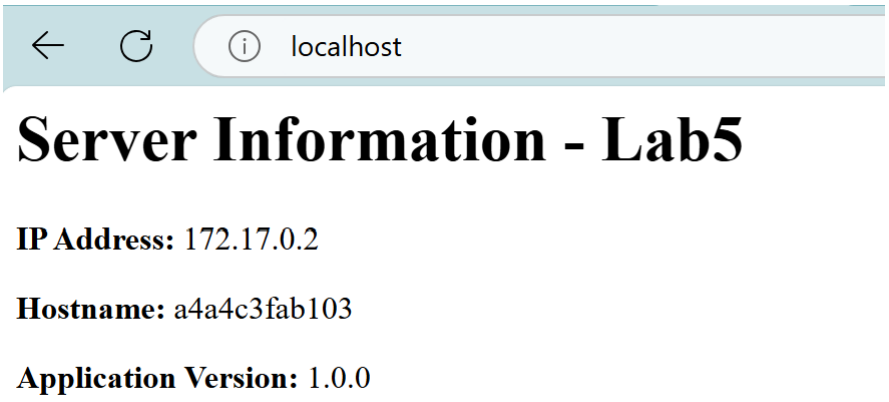
Rys. 8 Budowanie obrazu z nginx

Kontener działa prawidłowo. Pojawiają się logi których informacje oznaczają:

- **172.17.0.1** – Adres IP klienta
- **31/Mar/2025:18:14:46 +0000]** – Data i czas żądania (31 marca 2025, godzina 18:14:46 UTC).
- **"GET / HTTP/1.1"** – Żądanie HTTP (GET dla strony głównej /).
- **200** – Kod odpowiedzi HTTP.
- **217** – Liczba bajtów w odpowiedzi.
- **"-"** – Referer (brak wartości oznacza, że żądanie nie pochodziło z innej strony).
- **"Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/134.0.0.0 Safari/537.36 Edg/134.0.0.0"** – User-Agent, czyli przeglądarka klienta (w tym przypadku Microsoft Edge).



Rys. 9 Uruchomienie kontenera za pomocą portu 80 (port 8080 również działa)



Rys. 10 Przeglądarka: http://localhost:80