

Tworzenie obrazu kontenera dla prostej aplikacji

Marian Dorosz

Spis treści

1	Problem do rozwiązania	4
2	Rozwiązanie problemu	4
2.1	Kod aplikacji	4
2.2	Plik Dockerfile	5
2.2.1	FROM	5
2.2.2	WORKDIR	5
2.2.3	COPY	5
2.2.4	RUN	5
2.2.5	CMD	6
2.2.6	EXPOSE	6
2.3	Wynik polecenia docker build	6
2.4	Plik docker compose oraz wynik jego uruchomienia	7
2.5	Wynik polecenia curl.	7
3	Źródła	8

Spis rysunków

1	Kod aplikacji node.js	4
2	Kod znajdujący się w pliku Dockerfile	5
3	Wynik polecenia docker build	6
4	Zawartość pliku docker compose oraz wynik egzekucji	7
5	Wynik polecenia curl	7

1 Problem do rozwiązania

Należy zbudować **obraz kontenera** aplikacji napisanej przy pomocy **node.js**. Po zbudowaniu obrazu utworzyć plik typu **docker-compose**, którego zadaniem będzie uruchomienie kontenera z bazą mongoDB w wersji 3 oraz kontenera z aplikacją, który miał być utworzony. Po uruchomieniu pliku docker-compose sprawdzić poleceniem **curl** poprawność działania systemu. Po poprawnym zainicjowaniu kontenerów wynikiem polecenia **curl** powinien być napis **"Hello World!"**.

2 Rozwiązanie problemu

2.1 Kod aplikacji

A screenshot of a code editor showing a Node.js application. The code is written in JavaScript and is numbered from 1 to 20. It uses ES6 syntax with `require` and `var`. The application connects to a MongoDB instance, logs the connection status, and serves a 'Hello World!' response on port 3000.

```
1  var express = require('express');
2  var app = express();
3
4  var MongoClient = require('mongodb').MongoClient;
5  var assert = require('assert');
6
7  var url = 'mongodb://' + process.env.MONGO_IP + ':27017';
8  MongoClient.connect(url, function(err, db) {
9    assert.equal(null, err);
10   console.log("Connected correctly to server.");
11   // db.close();
12
13   app.get('/', function (req, res) {
14     res.send('Hello World!')
15   })
16
17   app.listen(3000, function () {
18     console.log('Example app listening on port 3000!')
19   })
20 })
```

Rysunek 1: Kod aplikacji node.js

2.2 Plik Dockerfile



```
1 FROM node:6.9.1
2
3 WORKDIR /app
4
5 COPY . .
6
7 RUN npm install
8
9 CMD ["node", "server.js"]
10
11 EXPOSE 3000
```

Rysunek 2: Kod znajdujący się w pliku Dockerfile

2.2.1 FROM

Instrukcja **From** oznacza na podstawie jakiego obrazu będzie budowany obraz kontenera. Jak widać na załączonym obrazku tworzony obraz będzie posiadać funkcjonalności kontenera node:6.9.1.

2.2.2 WORKDIR

Polecenie **WORKDIR** ustawia ścieżkę, w której będzie wykonywana każda komenda (np.: **RUN**, **CMD** itp.). Jeżeli dana ścieżka nie istnieje, to zostanie utworzona. Warto dodać, że wszystko będzie odbywać się wewnątrz kontenera. W tym przypadku będzie to folder **/app**.

2.2.3 COPY

Polecenie to kopiuje pliki z lokalnej maszyny do pamięci kontenera. W powyższym przypadku wszystkie pliki, które znajdują w folderze wraz z plikiem **Dockerfile** zostaną przekopiowane do folderu **/app**.

2.2.4 RUN

Polecenie **RUN** uruchamia polecenia podane jako argumenty w trakcie budowania kontenera. Wykonają się one jednorazowo. W przypadku tego kontenera zostało uruchomione polecenie **npm install**, które zainstaluje wszystkie zależności potrzebne do uruchomienia aplikacji.

2.2.5 CMD

Polecenie **CMD** przyjmuje jako argumenty komendy, które mają zostać wykonane przy każdym uruchomieniu kontenera. W tym wypadku będzie to uruchomienie skryptu o nazwie **server.js** przez aplikację **node**.

2.2.6 EXPOSE

EXPOSE to informacja dla osoby, która będzie używać kontenera. Przyjmuje ona jako argument numer portu oraz rodzaj protokołu. Ma ona sygnalizować jakie porty mają zostać wykorzystane w trakcie używania komendy **docker run**.

2.3 Wynik polecenia docker build

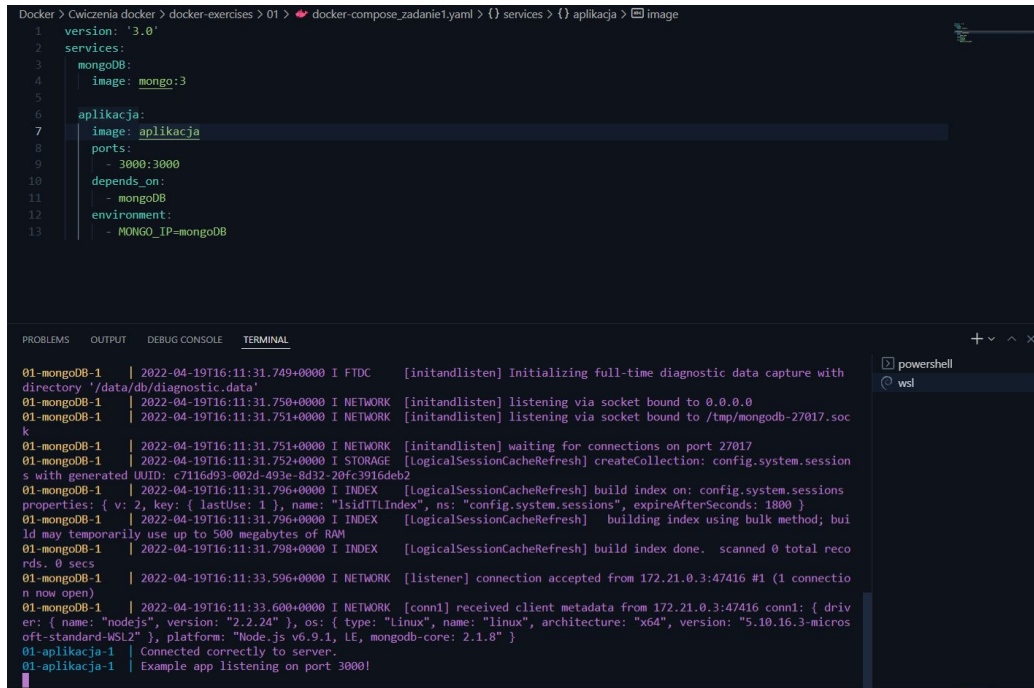
```
mario@DESKTOP-2JFCR00:/mnt/d/Programowanie/Docker/Cwiczenia docker/docker-exercises/01$ docker build -t aplikacja .
[+] Building 42.0s (9/9) FINISHED
=> [internal] load build definition from Dockerfile                                0.1s
=> => transferring dockerfile: 1488                                              0.1s
=> [internal] load .dockerignore                                                  0.0s
=> => transferring context: 28                                                    0.0s
=> [internal] load metadata for docker.io/library/node:6.9.1                    7.3s
=> [1/4] FROM docker.io/library/node:6.9.1@sha256:661a5a830a072c550ad8ad089d212867d0312c28f2992c01989f6c2789925f10  0.0s
=> [internal] load build context                                                  0.1s
=> => transferring context: 3.23kB                                                0.1s
=> CACHED [2/4] WORKDIR /app                                                      0.0s
=> [3/4] COPY . .                                                                0.1s
=> [4/4] RUN npm install                                                         33.2s
=> exporting to image                                                            1.0s
=> => exporting layers                                                            1.0s
=> => writing image sha256:e8193f5a4275cfd4304ad3e8c605768e124e591bc06b49627f2b7440f45c6ce4  0.0s
=> => naming to docker.io/library/aplikacja                                     0.0s

Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them
mario@DESKTOP-2JFCR00:/mnt/d/Programowanie/Docker/Cwiczenia docker/docker-exercises/01$
```

Rysunek 3: Wynik polecenia docker build

Polecenie **docker build** konstruuje obraz kontenera wykorzystując polecenia znajdujące się w pliku **Dockerfile**. Jak widać obraz został poprawnie utworzony, gdyż nie zostały zakomunikowane żadne błędy.

2.4 Plik docker compose oraz wynik jego uruchomienia



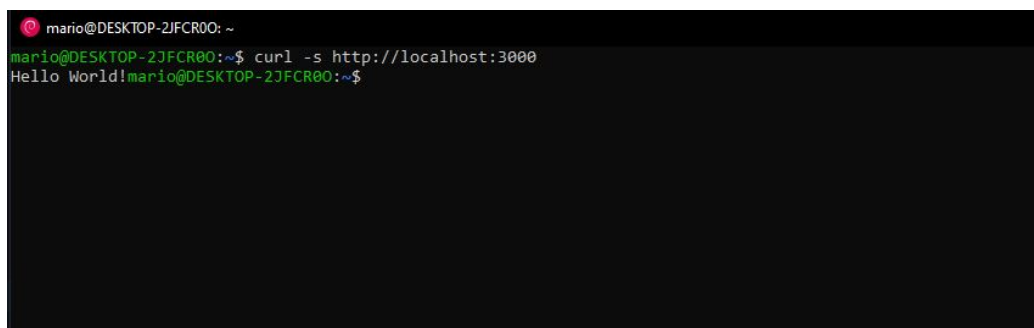
```
Docker > Ćwiczenia docker > docker-exercises > 01 > docker-compose_zadanie1.yaml > {} services > {} aplikacja > image
1 version: '3.0'
2 services:
3   mongoDB:
4     image: mongo:3
5
6   aplikacja:
7     image: aplikacja
8     ports:
9       - 3000:3000
10    depends_on:
11      - mongoDB
12    environment:
13      - MONGO_IP=mongoDB

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
01-mongoDB-1 | 2022-04-19T16:11:31.749+0000 I FTDC [initandlisten] Initializing full-time diagnostic data capture with
01-mongoDB-1 | 2022-04-19T16:11:31.750+0000 I NETWORK [initandlisten] listening via socket bound to 0.0.0.0
01-mongoDB-1 | 2022-04-19T16:11:31.751+0000 I NETWORK [initandlisten] listening via socket bound to /tmp/mongodb-27017.sock
01-mongoDB-1 | 2022-04-19T16:11:31.751+0000 I NETWORK [initandlisten] waiting for connections on port 27017
01-mongoDB-1 | 2022-04-19T16:11:31.752+0000 I STORAGE [LogicalSessionCacheRefresh] createCollection: config.system.sessions
01-mongoDB-1 | 2022-04-19T16:11:31.796+0000 I INDEX [LogicalSessionCacheRefresh] build index on: config.system.sessions
01-mongoDB-1 | 2022-04-19T16:11:31.796+0000 I INDEX [LogicalSessionCacheRefresh] building index using bulk method; bui
01-mongoDB-1 | 2022-04-19T16:11:31.798+0000 I INDEX [LogicalSessionCacheRefresh] build index done. scanned 0 total reco
01-mongoDB-1 | 2022-04-19T16:11:33.596+0000 I NETWORK [listener] connection accepted from 172.21.0.3:47416 #1 (1 connectio
01-mongoDB-1 | 2022-04-19T16:11:33.600+0000 I NETWORK [conn1] received client metadata from 172.21.0.3:47416 conn1: { driv
01-aplikacja-1 | Connected correctly to server.
01-aplikacja-1 | Example app listening on port 3000!
```

Rysunek 4: Zawartość pliku docker compose oraz wynik egzekucji

Jak widać na powyższym rysunku udało się uruchomić obie usługi, o czym świadczy komunikat **Connected correctly to server**.

2.5 Wynik polecenia curl.



```
mario@DESKTOP-2JFCR00: ~
mario@DESKTOP-2JFCR00:~$ curl -s http://localhost:3000
Hello World!mario@DESKTOP-2JFCR00:~$
```

Rysunek 5: Wynik polecenia curl

Jak widać wszystko zostało skonfigurowane poprawnie, ponieważ polecenie curl zwróciło napis **Hello World!**

3 Źródła

Autor aplikacji nodejs: github.com/Vizuri