COLLEGIUM WITELONA
Uczelnia Państwowa

Wydział Nauk Technicznych i Ekonomicznych
Kierunek Informatyka
Przedmiot Sieci Komputerowe

Sebastian Górski
nr alb. 43839

Sprawozdanie z Projektu – Docker/Sieci Komputerowe

Legnica 2024 rok

Spis treści

1 Wstęp

1.1 Opis projektu

Projekt miał na celu przygotowanie konfiguracji technologii Docker, która umożliwi łatwe uruchamianie aplikacji webowej „CN-Project". Aby osiągnąć oczekiwany efekt, konieczne było utworzenie kontenerów zawierających niezbędne usługi, takie jak: serwer aplikacji PHP z frameworkiem Laravel, baza danych MySQL, serwer Nginx oraz system cache Redis. Dzięki użyciu Dockera możliwe będzie zapewnienie jednorodnego środowiska pracy dla wszystkich członków zespołu, bez potrzeby ręcznego instalowania i konfigurowania aplikacji oraz jej zależności na każdym komputerze.

1.2 Podjęte kroki

Aby przygotować odpowiednie środowisko, zostały podjęte następujące kroki:

1. Zainstalowanie programu Docker na stacji roboczej.
2. Utworzenie pliku `docker-compose.yml` w folderze głównym projektu.
3. Utworzenie potrzebnych plików konfiguracyjnych.
4. Konfiguracja projektu.
5. Automatyzacja uruchomienia aplikacji.

1.3 Cele projektu

Główne cele wykonanego projektu to:

1. Szybkie i spójne środowisko deweloperskie: Dzięki Dockerowi każdy deweloper w zespole może uruchomić projekt na swoim komputerze w identycznym środowisku, eliminując problemy związane z różnicami w konfiguracji lokalnych maszyn.
2. Automatyzacja instalacji i konfiguracji: Projekt automatyzuje proces instalacji wszystkich niezbędnych komponentów aplikacji, takich jak PHP, MySQL, Redis oraz Nginx, co pozwala na oszczędność czasu przy wdrażaniu nowego środowiska.
3. Skalowalność i łatwość rozwoju: Użycie kontenerów pozwala na łatwe dodawanie nowych usług, takich jak Node.js czy NPM, w przyszłości.
4. Izolacja usług: Każda z usług działa w osobnym kontenerze, co ułatwia zarządzanie nimi niezależnie.

2 Instrukcja uruchomienia

2.1 Przed rozpoczęciem

Przed uruchomieniem projektu należy upewnić się, że posiadane są wymagane składniki:

- **Program Docker** - Jeżeli Docker nie jest zainstalowany na komputerze, należy go pobrać i zainstalować zgodnie z instrukcją na oficjalnej stronie: https://docs.docker.com/desktop/.
- **Aplikacja CN-Project** - Należy upewnić się, że projekt znajduje się na używanej maszynie i nic nie blokuje dostępu do niego. Jeżeli projekt nie znajduje się na lokalnym sprzęcie, należy go pobrać przy użyciu programu Git i komendy:

  ```
  git clone https://github.com/KarolZygadlo/CN-Project.git
  ```

2.2 Uruchomienie projektu

Aby uruchomić projekt „CN-Project" na swoim komputerze, należy wykonać następujące kroki:

1. Uruchomić program Docker Desktop na swoim systemie operacyjnym.
2. W konsoli systemu operacyjnego przejść do folderu, w którym znajduje się projekt (folder nazywa się `CN-Project`).
3. Projekt zostanie uruchomiony po wpisaniu i zatwierdzeniu poniższej komendy. Uruchomi ona kontenery w tle:

   ```
   docker compose up -d
   ```

   Flagi:

   - `-d` - oznacza uruchomienie kontenerów w trybie odłączonym (w tle), co pozwala na dalsze korzystanie z terminala.

   Jeżeli chcemy widzieć logi kontenerów na bieżąco, należy użyć tej samej komendy, ale bez flagi `-d`:

   ```
   docker compose up
   ```

   Uwaga: Proces uruchamiania może potrwać kilka minut, szczególnie przy pierwszym uruchomieniu, ponieważ Docker pobiera wszystkie niezbędne obrazy i zależności.
4. Po pobraniu i uruchomieniu projektu, aplikacja webowa będzie dostępna pod adresem `localhost`. Należy wpisać ten adres w przeglądarkę, aby wyświetlić stronę główną aplikacji.
5. Aby wyłączyć kontenery, należy wpisać w konsolę komendę:

   ```
   docker compose down
   ```

   Ta komenda zatrzyma wszystkie uruchomione kontenery.

2.3 Wejście do kontenera

W przypadku, gdy konieczne jest skorzystanie z narzędzi wewnątrz kontenera, najlepszym sposobem jest uruchomienie powłoki kontenera z poziomu maszyny lokalnej.

Poniższe kroki opisują sposób uruchomienia powłoki kontenera PHP.

1. Kontener musi być włączony. Można to sprawdzić przy pomocy komendy:

   ```
   docker ps
   ```

   Komenda ta pokazuje tylko aktualnie włączone kontenery.

   Poniżej przykład wyniku wywołania komendy `docker ps`:

   ```
   CONTAINER ID   IMAGE          COMMAND                  CREATED             STATUS         PORTS                                  NAMES
   0d9fb79953c5   nginx:latest   "/docker-entrypoint.…"   55 minutes ago      Up 6 minutes   0.0.0.0:80->80/tcp                     nginx-container
   5ea6707a3570   cn-project-php "environment/dev/php…"   55 minutes ago      Up 6 minutes   0/tcp, 9000/tcp                        php-container
   c344c89486ba   mysql:8.0      "docker-entrypoint.s…"   55 minutes ago      Up 6 minutes   0.0.0.0:3306->3306/tcp, 33060/tcp      mysql-container
   f3027f8193e4   redis:alpine   "docker-entrypoint.s…"   About an hour ago   Up 6 minutes   0.0.0.0:6379->6379/tcp                 redis-container
   ```

   Ostatnia kolumna wskazuje nazwy kontenerów — nazwa kontenera będzie używana do odwoływania się do niego.

2. Korzystając z nazwy kontenera (np. `php-container`), można uruchomić powłokę kontenera PHP przy użyciu komendy:

   ```
   docker exec -it php-container sh
   ```

   Elementy komendy:

   - `docker exec -flagi [nazwa kontenera] [polecenia]` - wykonuje wybrane polecenie w wybranym konenerze.
   - `-it` - Flaga `-i` oznacza „interaktywny” tryb, co pozwala na komunikację z kontenerem, a `-t` tworzy terminal, który umożliwia korzystanie z powłoki w trybie tekstowym.
   - `sh` - uruchamia powłokę systemu operacyjnego w kontenerze. Można używać innych powłok, takich jak `bash`, jeżeli są dostępne w kontenerze.

   Po zatwierdzeniu komendy użytkownik znajduje się wewnątrz kontenera i ma dostęp do wszystkich plików i programów, które się w nim znajdują.

3 Konfiguracja kontenerów

Plik `docker-compose.yml` to plik główny, wywoływany jako pierwszy przy użyciu technologii docker compose. Tutaj konfigurowane są wszystkie kontenery, sieci i wolumeny. Plik jest podzielony na 3 części:

- `services` - tutaj znajduje się szczegółowa konfiguracja wszystkich konterów, które zostają uruchomione poprzez docker compose. Konfiguracje opisane są w tak zwanych usługach (`service`). Sczegółowa analiza poszczególnych usług zostanie opisana w tym rozdziale.
- `volumes` - w tym miejscu znajduje się wolumen o nazwie *db-data*. Został utworzony, żeby dane były przechowywane w sposób trwały. Wolumen ten został wykorzystany w kontenerze bazy danych.
- `networks` - w tej sekcji tworzona jest sieć dockera, w której będą działały wszystkie kontenery z sekcji `services`. Jest to niezbędne do zapewnienia odpowiedniej łączności kontenerów między sobą. Sieć w tym projekcie nosi nazwę *cn-network* i jest skonfigurowany do używania sterownika `bridge`

Plik docker-compose jest plikiem w formacie YAML, którego struktura opiera się na wcięciach, a dane zapisywane są w formacie `klucz: wartość`

3.1 PHP

PHP jest serwerowym językiem programowania kluczowym dla działania aplikacji, w tym projekcie skonfigurowanym z frameworkiem `Laravel`, który usprawnia tworzenie aplikacji webowych. Użyto wersji PHP 8.2-fpm, zgodnej z Laravel 9 i nowszymi, zapewniającej wysoką wydajność dzięki FastCGI Process Manager (FPM).

Kontener PHP wzbogacono o dodatkowe zależności, takie jak Composer, zarządzający pakietami, oraz Redis, używany jako pamięć podręczna. Ten kontener obsługuje logikę aplikacji i komunikuje się z innymi usługami.

3.1.1 docker-compose.yml

Plik `docker-compose.yml` definiuje konfigurację kontenera PHP.

Listing 3.1: Konfiguracja kontenera php w pliku docker-compose.yml

```
1   php:
2       build:
3           context: .
4           dockerfile: /environment/dev/php/Dockerfile
5       container_name: php-container
6       working_dir: /var/www
7       volumes:
8           - ./:/var/www
9       expose:
10          - "9000:9000"
11      networks:
12          - cn-network
13      depends_on:
14          - mysql
15          - redis
```

W listingu 3.1 pokazana jest część pliku `docker-compose` odpowiedzialna za utworzenie kontenera PHP. Można zauważyć, że sekcja ta dzieli się na kilka części.

- `build` - komenda używana do tworzenia obrazu kontenera na podstawie plików źródłowych.
  - `context` - miejsce wywołania pliku (. - *aktualna ścieżka*).
  - `dockerfile` - lokalizacja pliku Dockerfile. W pliku Dockerfile został utworzony niestandardowy obraz na podstawie istniejącego już obrazu.
- `container_name` - nazwa kontenera - pole nieobowiązkowe, system sam może nadać nazwę, ale tutaj użytkownik posiada większą kontrolę.
- `working_dir` - określa katalog roboczy, w którym kontener rozpocznie pracę.
- `volumes` - określa jakie katalogi i pliki zostaną zamontowane do kontera. Wszystkie pliki źródłowe aplikacji zostają przeniesione do katalogu /var/www w kontenerze PHP.
- `expose` - określa porty, które kontener udostępnia innym kontenerom wewnątrz sieci (w tym przypadku inne kontenry będą mogły odnaleźć PHP na porcie 9000 w sieci cn-network).
- `networks` - sieci, w których kontener będzie pracował (w przypadku tego projektu wszystkie kontenery będą pracowały w jednej sieci, żeby miały do siebie swobodny dostęp).
- `depends_on` - komenda, w której można ustawić "priorytety" kontenerów. W tym przypadku kontener PHP zależy od kontenerów: `mysql` i `redis`, co oznacza, że PHP nie zostanie uruchomiony, dopóki wymienione kontenery nie zostaną uruchomione.

3.1.2 Dockerfile

Plik Dockerfile pozwala na utworzenie niestandardowego obrazu na podstawie obrazu pobranego z `Docker Hub`. Do tego pliku odwołała się usługa php w sekcji `build`. W pliku tym ważne jest wskazanie z jakiego obrazu bazowego należy korzystać.

Listing 3.2: Konfiguracja obrazu php w pliku Dockerilfe

```
FROM php:8.2-fpm

RUN apt-get update && apt-get install -y \
    git \
    unzip \
    zip \
    libpq-dev \
    libonig-dev \
    libcurl4-gnutls-dev \
    libxml2-dev \
    && docker-php-ext-install pdo pdo_mysql bcmath mbstring xml

RUN pecl install redis \
    && docker-php-ext-enable redis

COPY --from=composer:latest /usr/bin/composer /usr/bin/composer

WORKDIR /var/www

COPY .env.example .env
COPY . .

ENTRYPOINT ["environment/dev/php/entrypoint.sh"]
CMD ["php-fpm"]
```

W listingu 3.2 obrazem bazowym jest php w wersji 8.2-fpm. Wyznacza się go za pomocą komendy FROM. W późniejszych etap pracuje się na wybranym obrazie, modyfikując go w dowolny sposób.
W tym pliku kolejno:

1. Aktualizowane i instalowane są wymagane przez `Composer` biblioteki.
2. Instalowane jest rozszerzenie Redis, po czym zostaje aktywowany.
3. Kopiowany jest obraz najnowszej wersji `Composera` do tworzonego obrazu PHP.
4. Ustawiany jest katalog roboczy, dzięki czemu kolejne operacje będą w nim wykonywane.
5. Kopiowany jest plik .env.example do pliku .env (w pliku tym są wszystkie zmienne projektu)
6. Kopiowana jest cała zawartość programu.
7. Uruchamiany jest skrypt `entrypoint.sh`
8. Uruchamiany jest proces php-fpm, który będzie nasłuchwiał i obsługiwał żądania.

3.1.3 Skrypt Entrypoint

W projekcie, w celu instalacji `composera` i wstępnej inicjalizacji plików, uzyty został skrypt w języku `bash`.

Listing 3.3: Skrypt entrypoint.sh

```
#!/bin/bash

if [ ! -f .env ]; then
    cp .env.example .env
fi

if [ ! -d "vendor" ]; then
    composer install --no-dev --no-interaction --optimize-autoloader
fi

php artisan key:generate --force
php artisan migrate --force

exec php-fpm
```

Skrypt z listingu 3.3 wykonuje kilke kilka poleceń konsolowych w obrazie tworzonym w listingu 3.2. Na początku kopiowany jest plik ze zmiennymi, ponieważ `Laravel` potrzebuje niektórych danych w nim zawarych. Następnie instalowany jest composer. Po instalacji generowany jest unikalny klucz szyfrujący dla aplikacji Laravel, który zostaje zapisany w pliku `.env`. Następnie baza zostaje zmigrowana. Od teraz kontener PHP jest skofigurowany do pracy z frameworkiem `Laravel`.

3.2 MySQL

W projekcie został użyty obraz MySQL w wersji 8.0 jako silnik bazy danych. Zostało to podyktowane wysoką wydajnością, kompatybilnością z używanym środowiskiem oraz łatowścią korzystania i konfigurowania. Baza danych jest niezbędnym elementem każdej nowoczesnej aplikacji, w szczególności webowej.

3.2.1 docker-compose.yml

Cała konfiguracja serwera `MySQL` opiera się na 11 wierszach zapisanych w pliku `docker-copmpose`.

Listing 3.4: Konfiguracja kontenera MySQL w pliku docker-compose.yml

```
    mysql:
        image: mysql:8.0
        container_name: mysql-container
        ports:
            - "3306:3306"
        environment:
            - MYSQL_DATABASE=${DB_DATABASE}
            - MYSQL_ROOT_PASSWORD=${DB_PASSWORD}
        volumes:
            - db-data:/var/lib/mysql
        networks:
            - cn-network
```

W tym przypadku, w przeciwieństwie do konfiguracji PHP nie używamy klucza `build`, tylko `image`. Określamy w nim bezpośrednio obraz, z którego będziemy korzystać. Nowym kluczem jest również klucz `ports`. Jest to klucz porównywalny do `expose`, z taką różnicą, że port ten jest widoczny zewnątrz dockera - możemy np. dostać się do niego przez przeglądarkę. Dochodzi tutaj również klucz `environment`, który zawiera listę zmiennych. W konfiguracji użyta została tylko nazwa bazy danych i hasło do konta użytkowanika root. Dane te pobierane są z pliku .env, z pól, kórych nazwa jest wpisana w klamrach.

W kluczu environment można wpisać również wartości wprost, wtedy nie będą pobierane z pliku .env, tylko bezpośrednio z tekstu, który został wpisany jako wartość klucza.

Ważny jest również klucz `volumes`, w którym tworzymy wolumen łączący `db-data` z folderem mysql w kontenerze. Sposób tworzenia wolumenu jest ukazany w listingu 3.5

Listing 3.5: Wolumen db-data w pliku docker-compose.yml

```
volumes:
    db-data:
```

3.3 Nginx

Kolejnym kluczowym elementem aplikacji webowej jest serwer HTTP. Do obsługi ruchu HTTP użyty został serwer `Nginx`, który został skonfigurowany do prekazywania żądań do aplikacji PHP. Użyta została wersja `latest`, co oznacz najnowszą wersję z możliwych. Zabieg ten ma na celu ciągłe aktualizacje, które zwiększą bezpieczeństwo aplikacji. Mimo zmieniającej się wersji serwera `Nginx`, nie powinno tworzyć to problemów. Serwer Nginx działa na porcie 80, czyli na nim aplikacja będzie uruchamiana.

3.3.1 docker-compose.yml

W tej sekcji konfiguracja jest prosta. Wszystkie klucze zostały omówione w poprzednich podrozdziałach dotyczących innych usług.

Listing 3.6: Konfiguracja kontenera Nginx w pliku docker-compose.yml

```
    nginx:
        image: nginx:latest
        container_name: nginx-container
        ports:
            - "80:80"
        volumes:
            - ./environment/dev/nginx/nginx.conf:/etc/nginx/nginx.conf:ro
            - ./:/var/www
        networks:
            - cn-network
        depends_on:
            - php
```

Warto zwrócić uwagę na podpięcie pliku konfiguracyjnego do kontenera serwera HTTP. Znacznik :ro oznacza read-only (tylko do odczytu). Opis pliku znajduje się w rozdziale 3.3.2

3.3.2 nginx.conf

Plik `nginx.conf` definiuje w jaki sposób `Nginx` ma obsługiwać przychodzące żądania. W przypadku tego proejktu jest podzielony na dwie sekcje - `events` - listing 3.7 i `http` - listing 3.8.

Listing 3.7: Plik konfiguracyjny nginx.conf - sekcja events

```
events {
    worker_connections 1024;
}
```

W tym przypadku została określona liczba jednoczesnych połączeń na 1024.

Listing 3.8: Plik konfiguracyjny nginx.conf - sekcja http

```
http {
    server {
        listen 80;
        server_name localhost;

        root /var/www/public;
        index index.php index.html;

        location / {
            try_files $uri $uri/ /index.php?$query_string;
        }

        location ~ \.php$ {
            include fastcgi_params;
            fastcgi_pass php-container:9000;
            fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
            fastcgi_intercept_errors on;
        }

        location ~ /\.ht {
            deny all;
        }
    }
}
```

Ta sekcja jest bardziej rozbudowana. Mówi między innymi o tym, że serwer `Nginx` będzie nasłuchiwał na serwerze o nazwie localhost, na porcie 80. Katalog okreslony jako główny to /ver/www/public. Domyślnym plikiem indexowym jest plik `index` z rozszerzeniem `php` lub `html`.

Następnie jest `location /` - pole to określa w jaki sposób serwer ma obsłużyć żądanie HTTP wysłane na stronę. `Nginx` spróbuje znaleźć plik lub katalog odpowiadający żądanej ścieżce. Jeśli ścieżka nie istnieje, przekieruje żądanie do `index.php` wraz z parametrami `$query_string`.

Najdłuższa sekcja na liście to `location ~ \.php$` - określa ona regułę obsługi wszystkich plików PHP. Skonfigurowane zostały tutaj niezbędne parametry do komunikacji z PHP. Najbardziej interesującym nas elementem jest `fastcgi_pass` - ustawiamy tutaj, gdzie `Nginx` ma szukać serwera PHP *Wpisana musi być nazwa kontenera, na którym jest uruchomiony PHP wraz z portem!*

Na koniec zablokowany jest dostęp do wszystkich plików, kórych nazwa zaczyna się od .ht (np. `.htaccess`). Jest to istotne pod względem bezpieczeństwa aplikacji.

3.4 Redis

`Redis` to system bazodanowy `no-sql`, który działa w pamięci i przechowuje dane tymczasowe. Dzięki tej technologii zwiększa się wydajność aplikacji. Redis zostanie zintegrowany z kontenerem PHP i wspomoże dostęp do często używanych danych. Werjsa użyta w projekcie to `Alpine` - najnowsza, lżejsza wersja technologii.

3.4.1 docker-compose.yml

Listing 3.9: Konfiguracja kontenera Redis w pliku docker-compose.yml

```
    redis:
        image: redis:alpine
        container_name: redis-container
        command: redis-server --appendonly yes
        ports:
            - "6379:6379"
        networks:
            - cn-network
```

W listingu 3.9 widać konfigurację technologii `Redis`. Nowością tutaj jest klucz `command` z wartością `redis-server --appendonly yes`. Klucz ten wywołuje komendę wpisaną w wartości poczas uruchomienia kontenera. Wartość `redis-server` uruchamia serwer `Redis`. Parametr `--appendonly yes` włącza tryb zapisywania operacji do pliku.

4 Podsumowanie

W ramach projektu udało się stworzyć środowisko uruchomieniowe wykorzystując `Docker compose` dla aplikacji webowej opartej na technologiach takich jak: `PHP (Laravel), Nginx, MySQL i Redis`. Wykorzystana została do tego celu konfiguracja plikowa w plikach `docker-compose.yml, Dockerfile` itd.

Konfiguracja ta pozwala na uruchomienie środowiska programistycznego na dowolnym komputerze, który posiada pliki źródłowe i program Docker.

Początkowo pojawiły się problemy związane z nieznajomością używanych technologii. Przez długi czas był problem z połączeniem bazy danych, przez co na stronie głównej pojawiał się błąd z rodziny 5xx. Rozwiązaniem okazało się zmienienie podejścia budowania niestandardowego obrazu PHP z Laravelem i momentu kopiowania pliku `.env.example` do pliku `.env`. Pomogło również zminimalizowanie konfiguracji do korzystania z użytkownika `root` w konfiguracji `MySQL`.

Z niewiadomych przczyn pojawił się również problem z wykorzystaniem PHP w standardowej wersji. W tym przypadku zmienienie obrazu na wersję `FPM` oraz dodanie skryptu instalującego `Composera`, zamiast instalowanie go w pliku `Dockerfile` pomogło rozwiązać problem.

Środowisko mimo wszystko można jeszcze poprawić dodając nowe kontenery (np. `PHPMyAdmin` lub `Node.js`) oraz można popracować nad zwiększeniem wydajności.