Dokumentacja projektowa

Zarządzanie Systemami Informatycznymi

Projekt:

Konteneryzacja za pomocą Dockera aplikacji webowej stworzonej w Python Flask

Kierunek: Informatyka

Członkowie zespołu:
Piotr Olasik
Wojciech Grzywocz
Witold Pacholik

Gliwice, 2024/2025

Spis treści

1 Wprowadzenie

1.1 Cel projektu

Celem projektu jest wykorzystanie konteneryzacji za pomocą Dockera w stosunku do aplikacji webowej Pythona stworzonej przy użyciu pakietu Flask. Nasza aplikacja jest aplikacją webową służącą do sprawdzania pogody w podanym przez użytkownika miejscu na świecie. Projekt ma na celu praktyczne zademonstrowanie procesu tworzenia aplikacji webowej, oddzielenia logiki aplikacji od środowiska systemowego oraz zastosowania nowoczesnych rozwiązań do zarządzania środowiskami uruchomieniowymi. Jednym z założeń było umożliwienie łatwego wdrażania aplikacji na różnych platformach bez konieczności konfigurowania zależności lokalnie, co stanowi jedną z kluczowych korzyści wykorzystania Dockera w procesie wytwarzania i utrzymania oprogramowania. Projekt pozwala również zrozumieć podstawowe zasady działania kontenerów, obrazu Docker, zarządzania zależnościami oraz automatyzacji procesu budowania i uruchamiania aplikacji. Dzięki temu stanowi praktyczne wprowadzenie do zagadnień związanych z DevOps oraz infrastrukturą jako kodem (Infrastructure as Code, IaC), które są obecnie standardem w nowoczesnym cyklu życia oprogramowania. Dodatkowym aspektem projektu było uwzględnienie przenośności kodu źródłowego, tak aby możliwe było jego dalsze rozwijanie, testowanie i wdrażanie na serwerach lokalnych oraz w chmurze. Konteneryzacja zapewnia tym samym nie tylko uproszczony proces deploymentu, ale również zgodność środowiska uruchomieniowego na wszystkich etapach cyklu życia aplikacji.

2 Założenia projektowe

2.1 Założenia techniczne i nietechniczne

Założenia techniczne:

- Aplikacja webowa jest stworzona przy użyciu frameworka Flask (Python 3).
- Aplikacja umożliwia pobieranie aktualnych danych pogodowych dla wybranego miasta z publicznego API OpenWeatherMap.
- Dane pogodowe (miasto, temperatura, opis) są zapisywane do relacyjnej bazy danych PostgreSQL.
- Aplikacja obsługuje zarówno interakcję użytkownika (formularz HTML), jak i operacje na bazie danych (zapis, odczyt).
- Całość projektu jest uruchamiana w kontenerach Docker, z plikami konfiguracyjnymi:
  - `Dockerfile` – definiujący obraz kontenera aplikacji.
  - `docker-compose.yml` – konfiguracja kontenerów, w tym dla aplikacji Flask oraz bazy danych PostgreSQL.
- Komunikacja między kontenerami jest realizowana przy użyciu Docker Compose.
- Po uruchomieniu aplikacja będzie dostępna lokalnie przez przeglądarkę internetową.
- Dane dostępowe do bazy i API są przechowywane jako zmienne środowiskowe w pliku `.env`.

Założenia nietechniczne:

- Projekt ma charakter edukacyjny, dzięki niemu możemy zgłębić naszą wiedzę na temat konteneryzacji i procesu wytwarzania oprogramowania Służy zaprezentowaniu podstawowych właściwości Dockera i stanowi silne wprowadzenie do niego na bazie praktycznego przydatnego przykładu.
- Kod zawiera przejrzysty kod z rozdzieleniem logiki API, logiki bazodanowej i warstwy widoku.

- Dokumentacja ma pełnić rolę "przewodnika"dla użytkownika, umożliwiając zagłębienie się w idee projektu.
- Projekt nie zakłada skalowalności ani wysokiej dostępności – ma być zrozumiały, edukacyjny, przedstawiający wprowadzenie do Dockera na żywym przykładzie praktycznej aplikacji.

2.2 Stos technologiczny

W projekcie wykorzystano następujące technologie i narzędzia:

- **Język programowania:** Python 3.8
- **Framework webowy:** Flask
- **Zewnętrzne API:** OpenWeatherMap
- **Baza danych:** PostgreSQL
- **Sterownik bazy danych:** psycopg2
- **Platforma konteneryzacyjna:** Docker + Docker Compose
- **Zarządzanie zależnościami:** `requirements.txt`
- **Szablony HTML:** Jinja2 (`render_template`)
- **Edytor kodu (lokalnie):** PyCharm
- **Inne:** plik `.env` do zarządzania zmiennymi środowiskowymi

2.3 Oczekiwane rezultaty projektu

- Po zbudowaniu i uruchomieniu kontenerów, użytkownik może wprowadzić nazwę miasta i uzyskać dane pogodowe (temperatura, opis) z OpenWeatherMap API.
- Dane te zostają zapisane do bazy PostgreSQL jako historia zapytań.
- Użytkownik ma dostęp do aplikacji przez przeglądarkę internetową – aplikacja powinna być w pełni funkcjonalna lokalnie.
- Proces uruchamiania aplikacji jest zautomatyzowany przy użyciu plików `Dockerfile` oraz `docker-compose.yml`.
- Projekt powinien być możliwy do uruchomienia na dowolnym systemie wspierającym Dockera.

3 Realizacja projektu

3.1 Inicjalizacja bazy danych

Pierwszym krokiem było stworzenie bazy danych i tabeli przechowującej dane o pogodzie. W tym celu wykorzystano PostgreSQL. Tabela przechowywała identyfikator, nazwę miasta, temperaturę oraz opis pogody.

3.2 Połączenie z bazą danych

Do komunikacji z bazą danych użyto biblioteki `psycopg2`, która pozwalała na łatwe wykonanie zapytań SQL oraz zarządzanie połączeniami. W funkcji `get_conn()` definiowane było połączenie z bazą danych, przy użyciu zmiennych środowiskowych do określenia hosta, użytkownika, hasła i nazwy bazy danych.

```
def get_conn():
    return psycopg2.connect(
        host=os.getenv("DB_HOST", "db"),
        database=os.getenv("DB_NAME", "weatherdb"),
        user=os.getenv("DB_USER", "weatheruser"),
        password=os.getenv("DB_PASSWORD", "weatherpass")
    )
```

3.3 Funkcje zapisywania i pobierania danych

Aby zapisywać dane o pogodzie, stworzono funkcję `save_weather()`, która wstawiała dane o temperaturze i opisie pogody do bazy danych. W celu wyświetlenia ostatnich 10 wpisów z tabeli, użyto funkcji `get_all_weather()`, która zwracała dane w formie listy.

```python
def save_weather(city, temp, description):
    conn = get_conn()
    cur = conn.cursor()

    cur.execute( query: """
    INSERT INTO weather (city, temp, description)
    VALUES (%s, %s, %s);
    """, vars: (city, temp, description))

    conn.commit()
    cur.close()
    conn.close()

2 usages
def get_all_weather():
    conn = get_conn()
    cur = conn.cursor()

    cur.execute("""
    SELECT city, temp, description FROM weather ORDER BY id DESC LIMIT 10;
    """)

    weather_data = cur.fetchall()

    cur.close()
    conn.close()

    return weather_data
```

3.4 Integracja z API OpenWeather

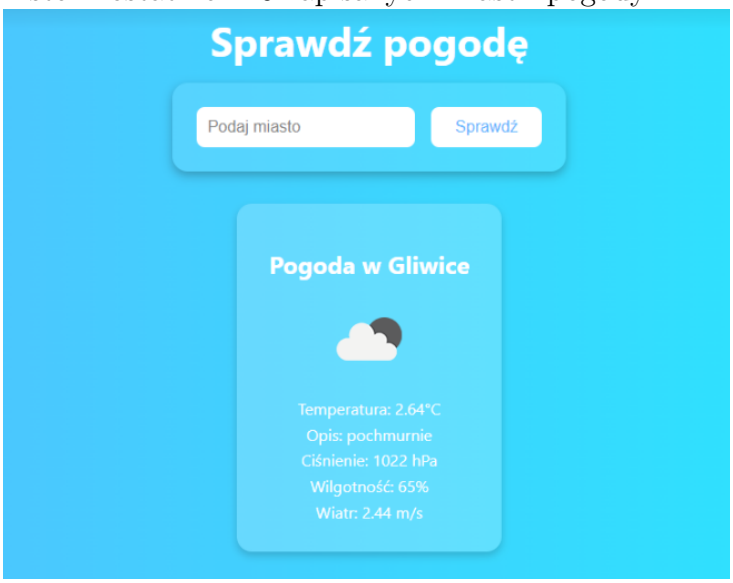
Zewnętrzne API OpenWeather było używane do pobierania aktualnych danych o pogodzie. Funkcja `get_weather()` przyjmowała nazwę miasta, wysyłała zapytanie do API i zwracała dane o temperaturze, ciśnieniu, wilgotności i prędkości wiatru.

```python
def get_weather(city):
    url = f"http://api.openweathermap.org/data/2.5/weather?q={city}&appid={API_KEY}&units=metric&lang=pl"
    response = requests.get(url)
    if response.status_code == 200:
        data = response.json()
        return {
            "city": city,
            "temp": data["main"]["temp"],
            "description": data["weather"][0]["description"],
            "pressure": data["main"]["pressure"],
            "humidity": data["main"]["humidity"],
            "wind_speed": data["wind"]["speed"],
            "icon": data["weather"][0]["icon"]
        }
    return None
```

3.5 Aplikacja webowa w Flask

Do stworzenia aplikacji webowej użyto frameworka Flask. Strona główna pozwalała na wpisanie nazwy miasta, po czym pobierane były dane o pogodzie, które były zapisywane w bazie danych. Użytkownik miał także dostęp do

historii ostatnich 10 zapisanych miast i pogody.



3.6 Skrypt do oczekiwania na PostgreSQL

Aby upewnić się, że baza danych była dostępna przed uruchomieniem aplikacji, zastosowano skrypt `wait-for-postgres.sh`. Skrypt ten czekał na dostępność serwera PostgreSQL, zanim aplikacja Flask rozpoczynała swoje działanie.

```
echo "Czekam na PostgreSQL..."
until pg_isready -h db -p 5432; do
  sleep 1
done

echo "PostgreSQL dostępny - startuję aplikację!"
exec python main.py
```

3.7 Dockerfile i konfiguracja środowiska

W projekcie użyto Dockera do konteneryzacji aplikacji. Dockerfile został stworzony, aby zbudować obraz aplikacji Python 3.11, zainstalować wymagane zależności, a także skonfigurować połączenie z bazą danych PostgreSQL.

```
FROM python:3.11

WORKDIR /app

COPY requirements.txt .
RUN apt-get update && apt-get install -y postgresql-client && pip install -r requirements.txt

COPY app/ .

RUN chmod +x wait-for-postgres.sh

CMD ["./wait-for-postgres.sh"]
```

3.8 Uruchomienie programu:

Aby uruchomić program należy:

- 1. Otworzyć CMD w folderze projektu.
- 2. Wpisać: docker compose up –build
- 3. Wejść na wyświetlony link http

Po zakończeniu działania z programem można wpisać: docker compose down

4 Wnioski

- *Spostrzeżenia*
  Projekt pozwolił na zrozumienie, jak integracja różnych technologii (takich jak Flask, PostgreSQL i API) może współdziałać w praktycznym rozwiązaniu. Kluczowym elementem było zarządzanie danymi o pogodzie oraz ich zapisywanie w bazie danych, co okazało się być łatwe do wdrożenia przy pomocy odpowiednich bibliotek. Dodatkowo, dzięki zastosowaniu Dockera, proces uruchamiania aplikacji stał się bardziej elastyczny i przenośny.

- *Osiągnięcia*
  Projekt udało się zrealizować w całości, co obejmowało stworzenie aplikacji webowej do wyświetlania danych pogodowych, połączenie z zewnętrznym API w celu pobierania danych oraz implementację bazy danych w PostgreSQL. Dodatkowo udało się zoptymalizować proces uruchamiania aplikacji za pomocą Dockera i skryptów przygotowujących środowisko. Aplikacja działa stabilnie i umożliwia użytkownikowi interakcję z danymi o pogodzie. Pozwoliła nam również zgłębić wiedzę na temat Dockera, konteneryzacji i ich praktycznego wykorzystania.

- *Potencjał rozwoju*
  Projekt ma duży potencjał rozwoju. Można rozbudować aplikację o dodatkowe funkcje, takie jak prognoza pogody na kilka dni do przodu, możliwość pobierania danych pogodowych dla wielu miast jednocześnie, czy dodanie mapy z lokalizacjami miast. Istnieje także możliwość integracji z innymi źródłami danych pogodowych oraz udostępniania aplikacji jako usługę w chmurze, co zwiększyłoby jej dostępność i skalowalność.

5 Bibliografia, źródła

1. https://docs.docker.com/manuals/
2. https://www.tutorialspoint.com/flask/index.htm
3. https://www.tutorialspoint.com/python/index.htm
4. https://docs.python.org/3/
5. https://openweathermap.org/api
6. https://openweathermap.org/api/one-call-3