Autor: Hanna Radchenko

Przedmiot: Narzędzia i metody inżynierii oprogramowania

Data: 10.12.2025

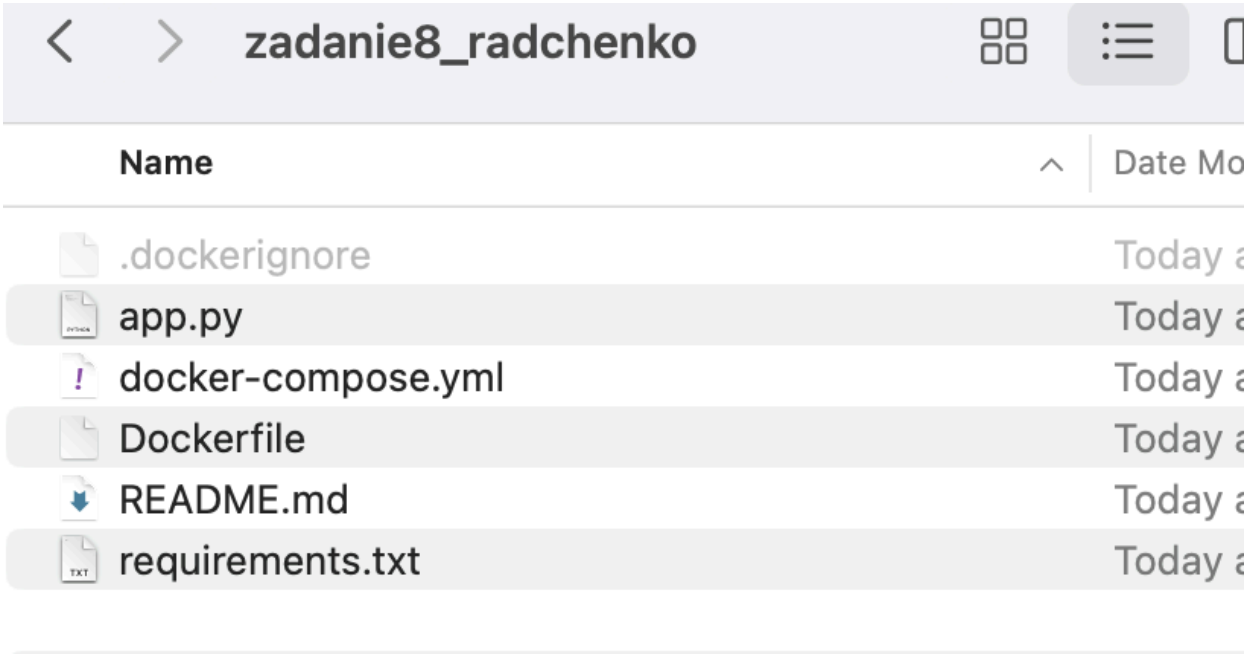
1. Wprowadzenie

Celem zadania było przygotowanie prostej aplikacji API w Pythonie, umieszczenie jej w kontenerze Dockera oraz uruchomienie za pomocą narzędzia `docker-compose`. Efektem końcowym jest działający endpoint HTTP, zwracający odpowiedź JSON.

Projekt wykorzystuje:

- Flask jako framework do budowy API
- Gunicorn jako produkcyjny serwer aplikacji
- Docker i Docker Compose do konteneryzacji i automatycznego uruchamiania środowiska
- Multi-stage Dockerfile do optymalizacji obrazu

2. Struktura projektu



3. Kod aplikacji (API)

Aplikacja została napisana w Pythonie z użyciem frameworka Flask. Serwer zwraca prosty komunikat JSON pod adresem.

Kod tworzy instancję aplikacji Flask i definiuje jedną ścieżkę – `/`.

Po jej wywołaniu serwer zwraca odpowiedź JSON za pomocą funkcji `jsonify()`.

Kod:

app.py 1 × docker-compose.yml

Users > hannaradcenko > Documents > zadanie8_radchenko > app.py > ...

```python
from flask import Flask, jsonify, request

app = Flask(__name__)

@app.get("/")
def hello():
    return jsonify({"message": "Hello from Docker API!", "status": "running"})

@app.get("/hello/<name>")
def hello_name(name):
    return jsonify({"message": f"Hello, {name}!"})

@app.get("/items")
def get_items():
    items = ["apple", "banana", "orange"]
    return jsonify({"items": items})

@app.post("/add")
def add_item():
    data = request.get_json()
    return jsonify({
        "received": data,
        "info": "Data successfully processed!"
    })

```

Dostępne endpointy:

Endpoint	Metoda	Opis
/	GET	Zwraca podstawowy komunikat JSON z informacją, że API działa.
/hello/<name>	GET	Zwraca spersonalizowany komunikat powitalny dla podanego imienia.

/items	GET	Zwraca listę przykładowych przedmiotów w formacie JSON.
/add	POST	Przyjmuje dane JSON w treści żądania i zwraca potwierdzenie ich odebrania.

4. Dockerfile

Dockerfile zawiera instrukcje do zbudowania obrazu Dockera z aplikacją Pythonową. Wykorzystuje oficjalny obraz `python:3.9-slim`, kopiuje pliki projektu, instaluje wymagane zależności i przygotowuje serwer Gunicorn.

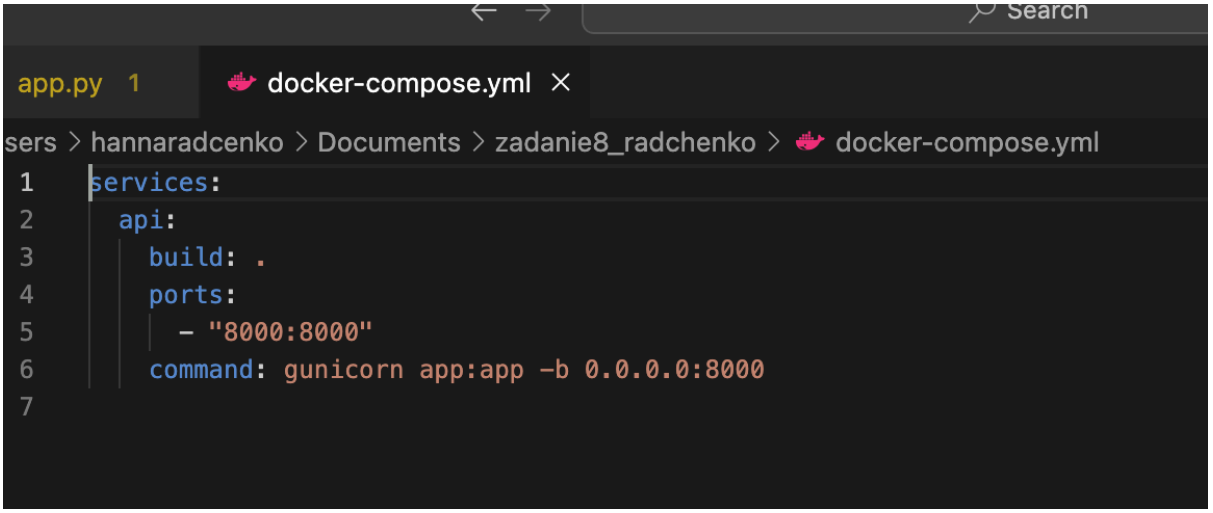
Dockerfile

```
# Stage 1: Builder
FROM python:3.11-slim AS builder

WORKDIR /app

# Zainstaluj wymagane narzędzia systemowe
RUN apt-get update && apt-get install -y \
    build-essential \
    libssl-dev \
    libffi-dev \
    && rm -rf /var/lib/apt/lists/*

COPY requirements.txt .

# Instalacja pakietów Python dla użytkownika
RUN pip install --user --no-cache-dir -r requirements.txt

# Stage 2: Final
FROM python:3.11-slim

WORKDIR /app

COPY --from=builder /root/.local /root/.local
COPY . .

ENV PATH=/root/.local/bin:$PATH
EXPOSE 8000

# Uruchomienie produkcyjnego WSGI
CMD ["gunicorn", "-b", "0.0.0.0:8000", "app:app"]
```

5. Plik docker-compose.yml

Plik `docker-compose.yml` definiuje usługę o nazwie `api`.

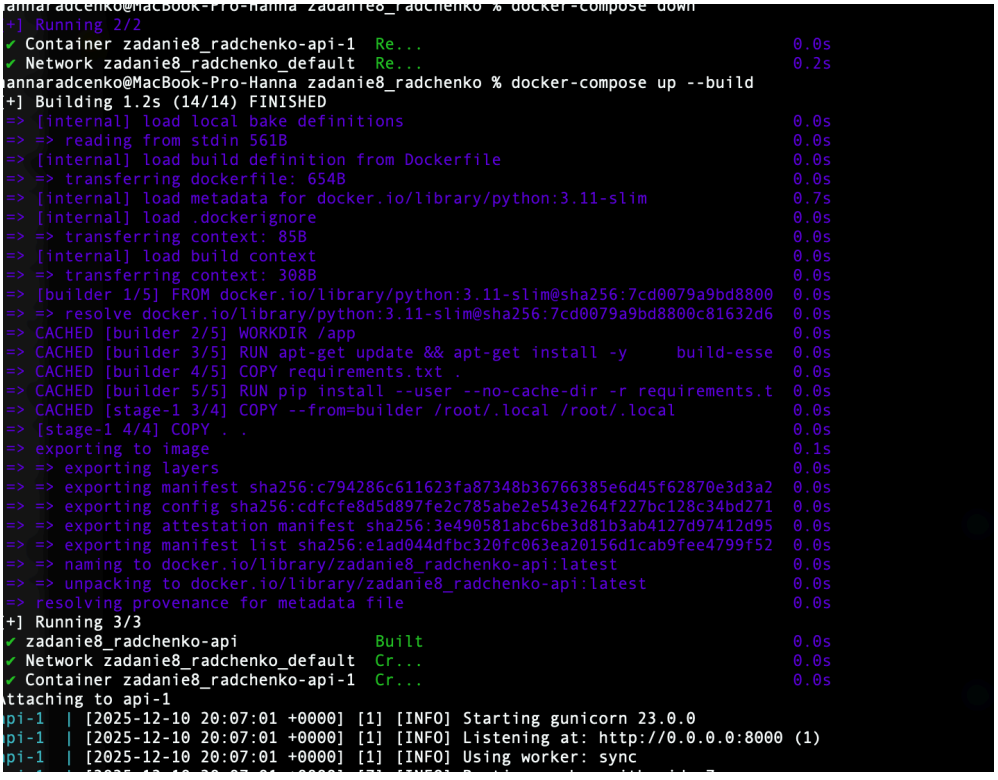
Określa porty oraz sposób budowania kontenera.



6. Uruchamianie aplikacji

Aby uruchomić aplikację, należy wykonać:

```
docker-compose up --build
```

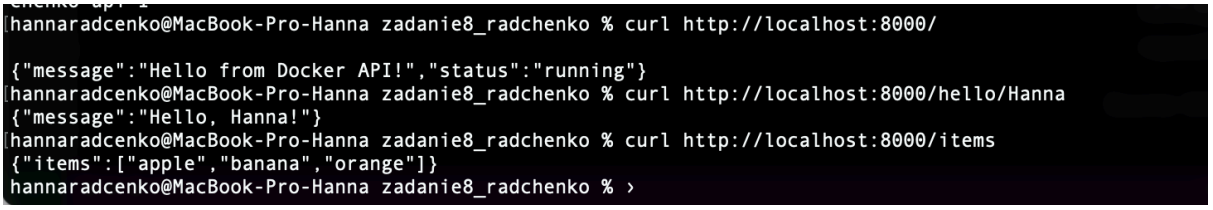


Uruchomienie w tle:

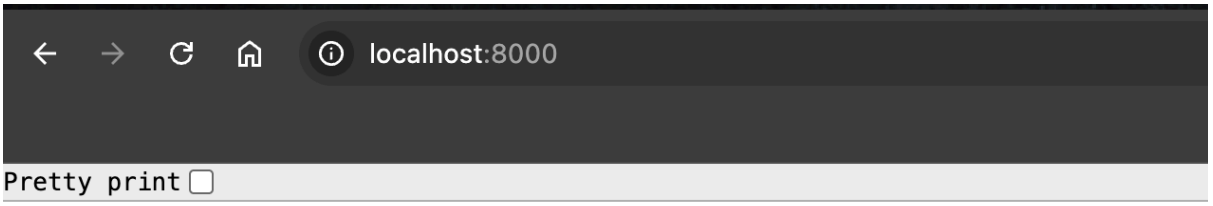
```
docker-compose up -d
```

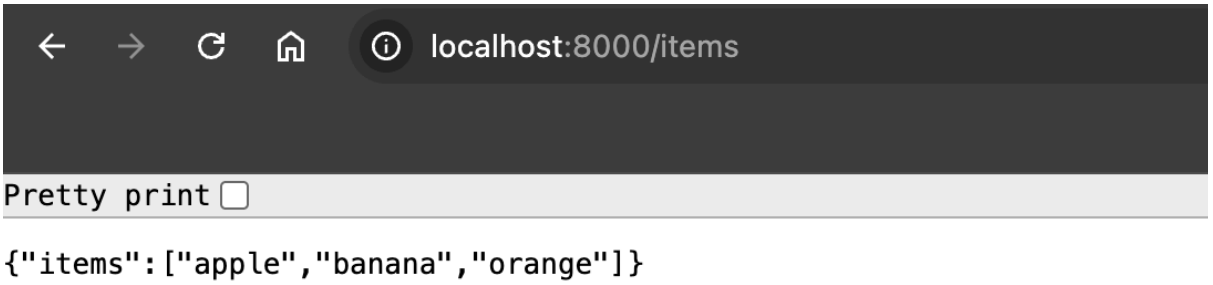
Sprawdzenie działania:

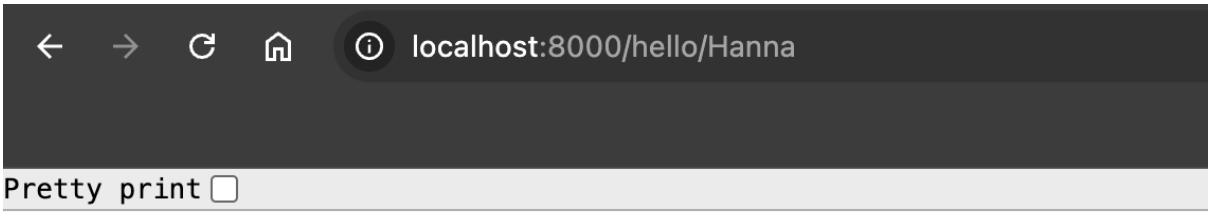
```
curl http://localhost:8000/
```



7. Test połączenia z API







{"message":"Hello, Hanna!"}

Wszytko działa.