
	<p>Politechnika Bydgoska im. Jana i Jędrzeja Śniadeckich</p> <p>Wydział Telekomunikacji, Informatyki i Elektrotechniki</p> <p><b>Nowoczesne Technologie Przetwarzania Danych</b></p> <p>Laboratorium 04</p> <p>Temat: Docker i konteneryzacja modelu ML</p>	
---	---	---

## Cel ćwiczenia

Celem ćwiczenia jest tworzenie obrazu Dockera z modelem ML; uruchamianie kontenera i testowanie endpointów; konfiguracja Docker Compose dla aplikacji ML.

## Zadanie 1: Przygotowanie aplikacji API

Utworzono plik app.py, który posiada konfigurację z poprzednich zajęć oraz plik requirements.txt który zawiera wszystkie niezbędne biblioteki.

```
(.venv) PS C:\Users\Aga\PycharmProjects\Sprawka> pip install -r C:\Users\Aga\PycharmProjects\Sprawka\lab4_NTPD\requirements.txt
Collecting flask==2.0.1 (from -r C:\Users\Aga\PycharmProjects\Sprawka\lab4_NTPD\requirements.txt (line 1))
  Obtaining dependency information for flask==2.0.1 from https://files.pythonhosted.org/packages/54/4f/1b294c1a4ab7b2ad5ca5fc4a9a05a22ef1ac48be126289d97668852d4ab3/Flask-2.0.1-py3-none-any.whl.metadata
  Downloading Flask-2.0.1-py3-none-any.whl.metadata (3.8 kB)
Collecting scikit-learn==0.24.2 (from -r C:\Users\Aga\PycharmProjects\Sprawka\lab4_NTPD\requirements.txt (line 2))
  Downloading scikit-learn-0.24.2.tar.gz (7.5 MB)
 7.5/7.5 MB 2.4 MB/s eta 0:00:00
Installing build dependencies ... done
```

## Zadanie 2: Dockerfile i budowa obrazu

W tym celu utworzono plik dockerfile znajdujący się w repozytorium, który posiada wszystkie wytyczne podane w zadaniu.

```
(.venv) PS C:\Users\Aga\PycharmProjects\Sprawka\lab4_ntpd> docker build -t lab4_ntpd .
[+] Building 11s.4s (10/10) FINISHED
=> [internal] load build definition from Dockerfile
=> == transferring Dockerfile: 228B
=> [internal] load metadata for docker.io/library/python:3.9-slim
=> [internal] load .dockerignore
=> == transferring context: 2B
=> [1/5] FROM docker.io/library/python:3.9-slim@sha256:e52ca5f579cc58fed41efc0b55aded5d0ccfec7a15c6a70acfb4ab42fc19ad00
=> == resolve docker.io/library/python:3.9-slim@sha256:e52ca5f579cc58fed41efc0b55aded5d0ccfec7a15c6a70acfb4ab42fc19ad00
=> == sha256:9a5d17a74709c4e5d0ff58220d540d205797a080e4d8fccc30f1e0210f 24MB / 24MB
=> == sha256:da1cb0d584f421af08250ce9a3d0a0301aa331260700070ac42c23da420936a 14.93MB / 14.93MB
=> == sha256:ce04908de99f5777ee485402f493307ae87ec374f4ee181d089117531a21 3.51MB / 3.51MB
=> == sha256:ae999acdb790c3a199909cfc795fdaa24ad0a04ab27b5c48e2b754b2c5fad 28.20MB / 28.20MB
=> == extracting sha256:ae999acdb790c3a199909cfc795fdaa24ad0a04ab27b5c48e2b754b2c5fad
=> == extracting sha256:ce04908de99f5777ee485402f493307ae87ec374f4ee181d089117531a21
=> == extracting sha256:da1cb0d584f421af08250ce9a3d0a0301aa331260700070ac42c23da420936a
=> == extracting sha256:9a5d17a74709c4e5d0ff58220d540d205797a080e4d8fccc30f1e0210f
=> [internal] load build context
=> == transferring context: 1.41kB
=> [2/5] WORKDIR /app
=> [3/5] COPY requirements.txt
=> [4/5] RUN pip install --no-cache-dir -r requirements.txt
=> [5/5] CMD
```

## Zadanie 3: Uruchamianie kontenera i testowanie endpointu

Poniżej znajdują się screeny z uruchomienia kontenera oraz testowania endpointu

```
(.venv) PS C:\Users\Aga\PycharmProjects\Sprawka\lab4_ntpd> docker run -p 5000:5000 lab4_ntpd
[2025-04-02 15:33:18 +0000] [1] [INFO] Starting gunicorn 20.1.0
[2025-04-02 15:33:18 +0000] [1] [INFO] Listening at: http://0.0.0.0:5000 (1)
[2025-04-02 15:33:18 +0000] [1] [INFO] Using worker: sync
[2025-04-02 15:33:18 +0000] [7] [INFO] Booting worker with pid: 7
```

Ponownie był problem z cURL, więc należało zmodyfikować komendy. Wynik tych operacji wygląda następująco:

```
(.venv) PS C:\Users\Aga\PycharmProjects\Sprawka\lab4_ntpd> curl -X POST http://localhost:5000/predict `
>> -Method POST `
>> -Headers @{"Content-Type"="application/json"} `
>> -Body '{"values": [1, 2, 3]}'

StatusCode      : 200
StatusDescription : OK
Content          : {
  "prediction": [
    1.1999999999999997,
    1.9999999999999998,
    2.8
  ]
}

RawContent      : HTTP/1.1 200 OK
                  Connection: close
                  Content-Length: 82
                  Content-Type: application/json
                  Date: Wed, 02 Apr 2025 15:36:33 GMT
                  Server: Werkzeug/3.1.3 Python/3.13.2

                  {
                    "prediction": [
                      1.199999999...
                    ]
                  }

Forms           : {}
Headers         : {[Connection, close], [Content-Length, 82], [Content-Type, application/json], [Date, Wed, 02 Apr 2025 15:36:33 GMT]...}
Images          : {}
InputFields     : {}
Links           : {}
ParsedHtml      : mshtml.HTMLDocumentClass
RawContentLength : 82
```

## Zadanie 4: Konfiguracja Docker Compose

Utworzono plik docker-compose.yml, którego kod znajduje się w repozytorium

## Zadanie 5: Uruchomienie aplikacji w trybie produkcyjnym

W pliku readme jest instrukcja jak uruchomić aplikację.

Zmienne środowiskowe:

Zmienna środowiskowa	Domyślna wartość	Opis
FLASK_ENV	production	Określa środowisko pracy (development/production). W trybie development Flask uruchamia debugger.
FLASK_APP	app.py	Główny plik aplikacji Flask.
PORT	5000	Port, na którym działa serwer.
MODEL_TYPE	LinearRegression	Typ modelu ML (możliwość zmiany w przyszłości).
REDIS_URL	redis://redis:6379/0	Adres URL Redis (używany w Docker Compose).

Przykład użycia:

```
$env:FLASK_ENV="development"
```

```
$env:PORT=8000
```

```
python app.py
```

Wymagania zasobowe:

Zasób	Minimalne	Zalecane
<b>CPU</b>	1 core	2 cores
<b>RAM</b>	512 MB	1 GB
<b>Dysk</b>	100 MB	500 MB
<b>System</b>	Windows/Linux/macOS	Docker (w przypadku konteneryzacji)

## Wnioski

Aplikacja jest **łatwa w konfiguracji**, **mało zasobożerna** i gotowa do wdrożenia zarówno na lokalnych maszynach, jak i w chmurze (np. AWS/Azure). Użycie Docker Compose dodatkowo upraszcza zarządzanie zależnościami (np. Redis). W przyszłości można dodać więcej zmiennych środowiskowych (np. do konfiguracji modelu ML) lub rozszerzyć monitoring (np. dodając Prometheus).