POLITECHNIKA
LUBELSKA

POLITECHNIKA
LUBELSKA
WYDZIAŁ ELEKTROTECHNIKI
I INFORMATYKI

Programowanie Full-Stack w Chmurze Obliczeniowej

Sprawozdanie 2

Sebastian Słupny

Prowadzący

Sławomir Przyłucki

1. (max. 100%)

Wykorzystując opracowaną aplikację (kod + Dockerfile) z zadania nr1 należy:

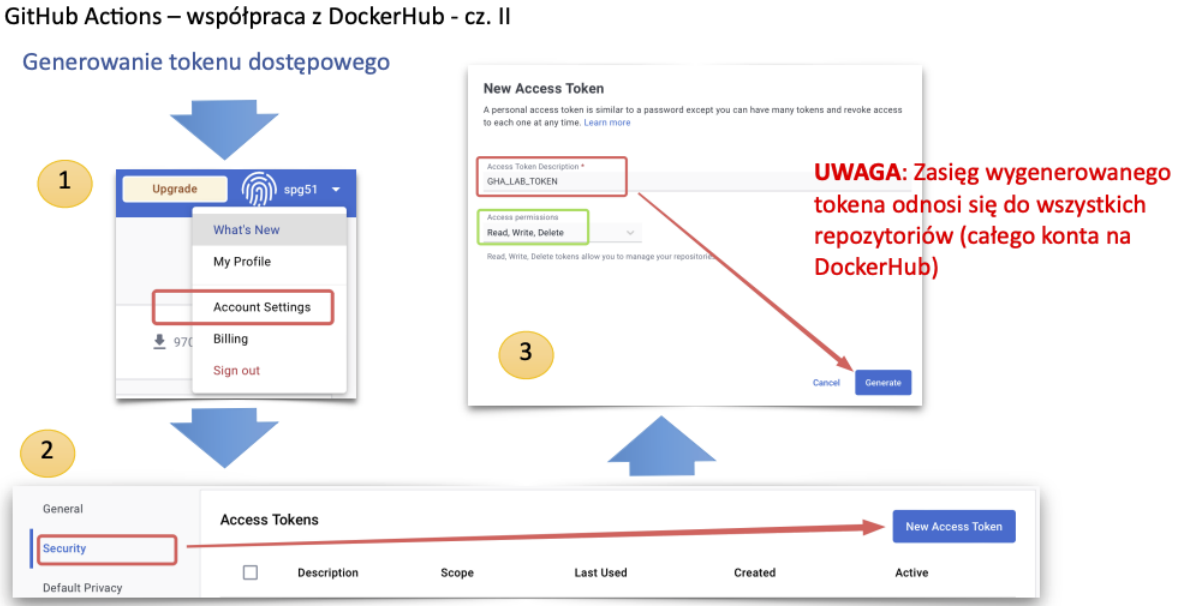
a. zbudować, uruchomić i potwierdzić poprawność działania łańcucha Github Actions, który zbuduje obrazy kontenera z tą aplikacją na architektury: linux/arm64/v8 oraz linux/amd64 wykorzystując QEMU

Aby wykonać zadanie opracowałem plik konfiguracyjny build.yaml, który opisuje wszystkie kroki do wykonania.

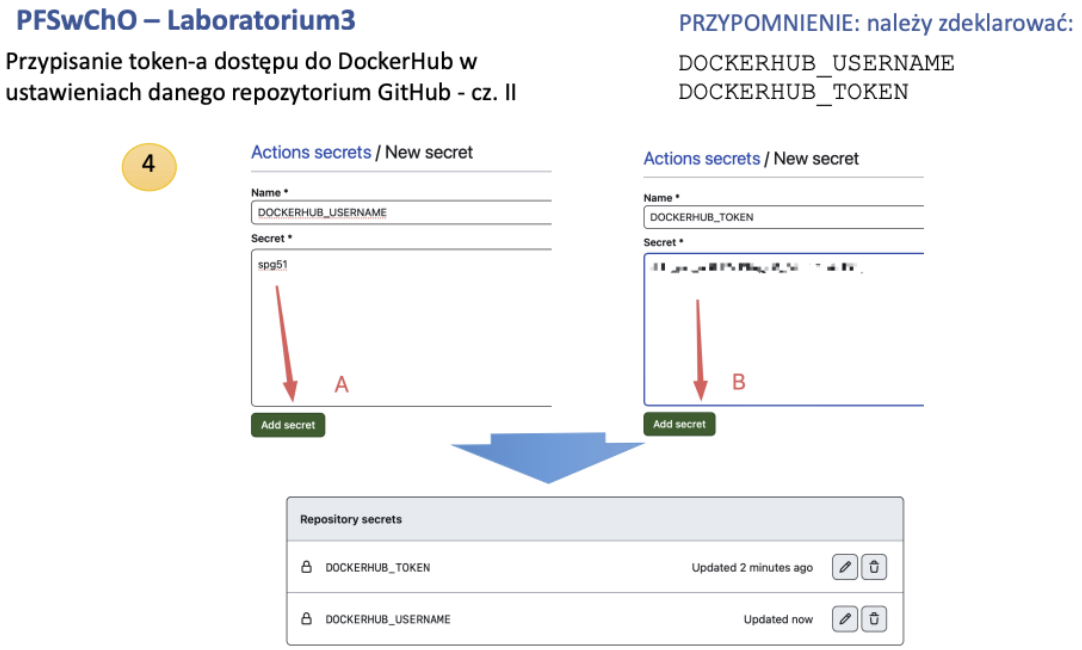
.github > workflows > ! build.yaml

```yaml
name: Sebastian Slupny Zadanie 2 Build and Push Docker Images

on:
  push:
    branches:
      - main

jobs:
  build:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout repository
        uses: actions/checkout@v3

      - name: Set up QEMU
        uses: docker/setup-qemu-action@v2

      - name: Set up Docker Buildx
        uses: docker/setup-buildx-action@v2

      - name: Login to Docker Hub
        uses: docker/login-action@v2
        with:
          username: ${{ secrets.DOCKER_USERNAME }}
          password: ${{ secrets.DOCKER_PASSWORD }}

      - name: Build and push Docker image (linux/amd64)
        uses: docker/build-push-action@v3
        with:
          context: .
          push: true
          tags: |
            sebastianslupny/pwcho_zad1:amd64
          platforms: linux/amd64

      - name: Build and push Docker image (linux/arm64/v8)
        uses: docker/build-push-action@v3
        with:
          context: .
          push: true
          tags: |
            sebastianslupny/pwcho_zad1:arm64
          platforms: linux/arm64/v8
```

Do logowania wykorzystano zmienne tajne



Zrzut instrukcji prowadzącego zajęcia



Zrzut instrukcji prowadzącego zajęcia

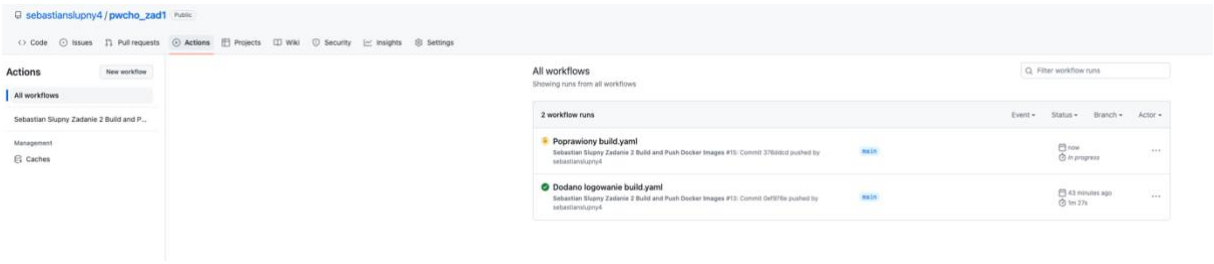
Po skonfigurowaniu wszystkiego wykonano commit i push do repozytorium.

```
sebastian_slupny@macbook-pro-sebastian PWCHO % git add .
sebastian_slupny@macbook-pro-sebastian PWCHO % git commit -m "Dodano logowanie build.yaml"
[main 0ef976e] Dodano logowanie build.yaml
 1 file changed, 2 insertions(+), 2 deletions(-)
sebastian_slupny@macbook-pro-sebastian PWCHO % git push
Enumerating objects: 9, done.
Counting objects: 100% (9/9), done.
Delta compression using up to 16 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (5/5), 412 bytes | 412.00 KiB/s, done.
Total 5 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/sebastianslupny4/pwcho_zad1
   3e222e0..0ef976e  main -> main
sebastian_slupny@macbook-pro-sebastian PWCHO %
```

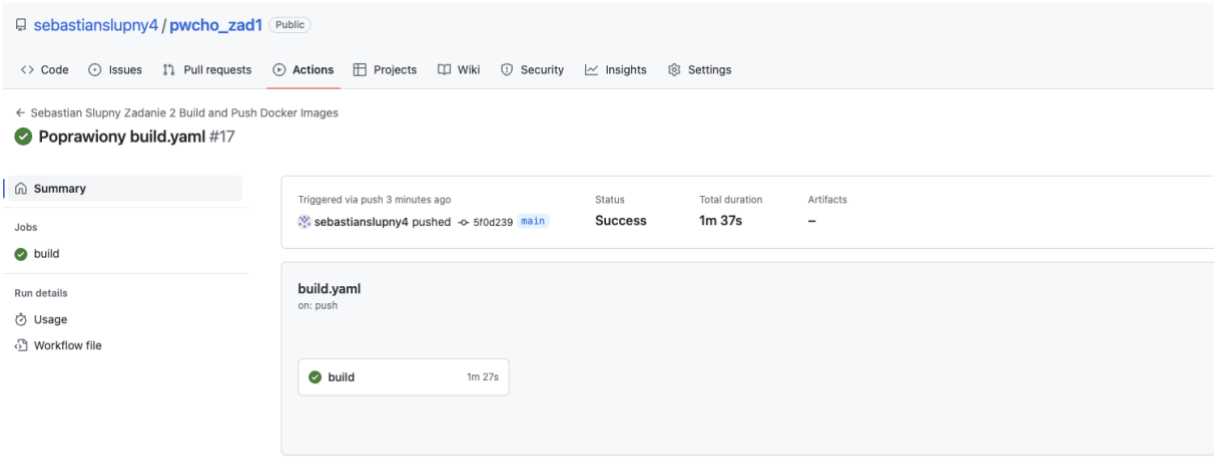
Zrzut ekranu z terminala

Wrzuciłem do repozytorium, następnie łańcuch został uruchomiony automatycznie.

Stan możemy sprawidzić w repo github w zakładce Actions.

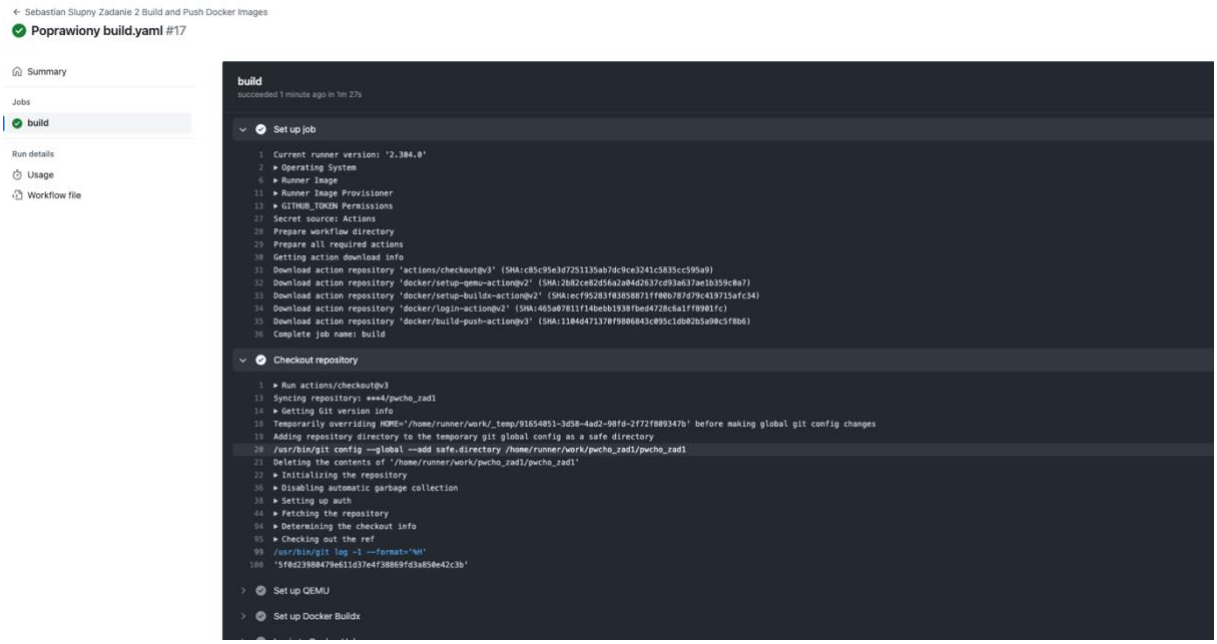


Zrzut Actions w tracie budowania

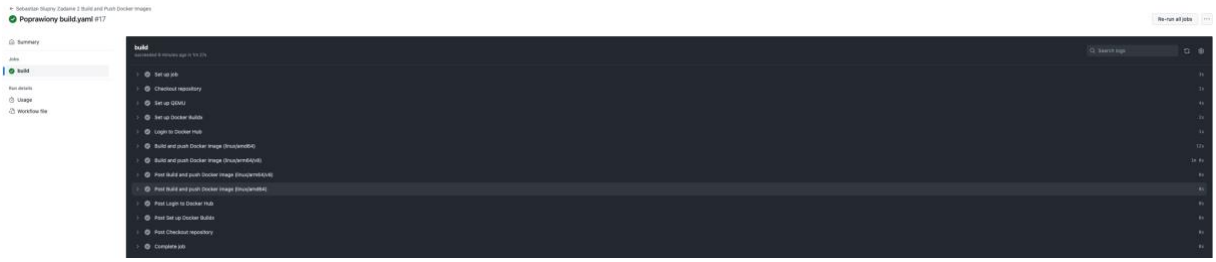


Zrzut Actions po zbudowaniu

Aby sprawdzić wykonanie możemy zobaczyć stan workflow



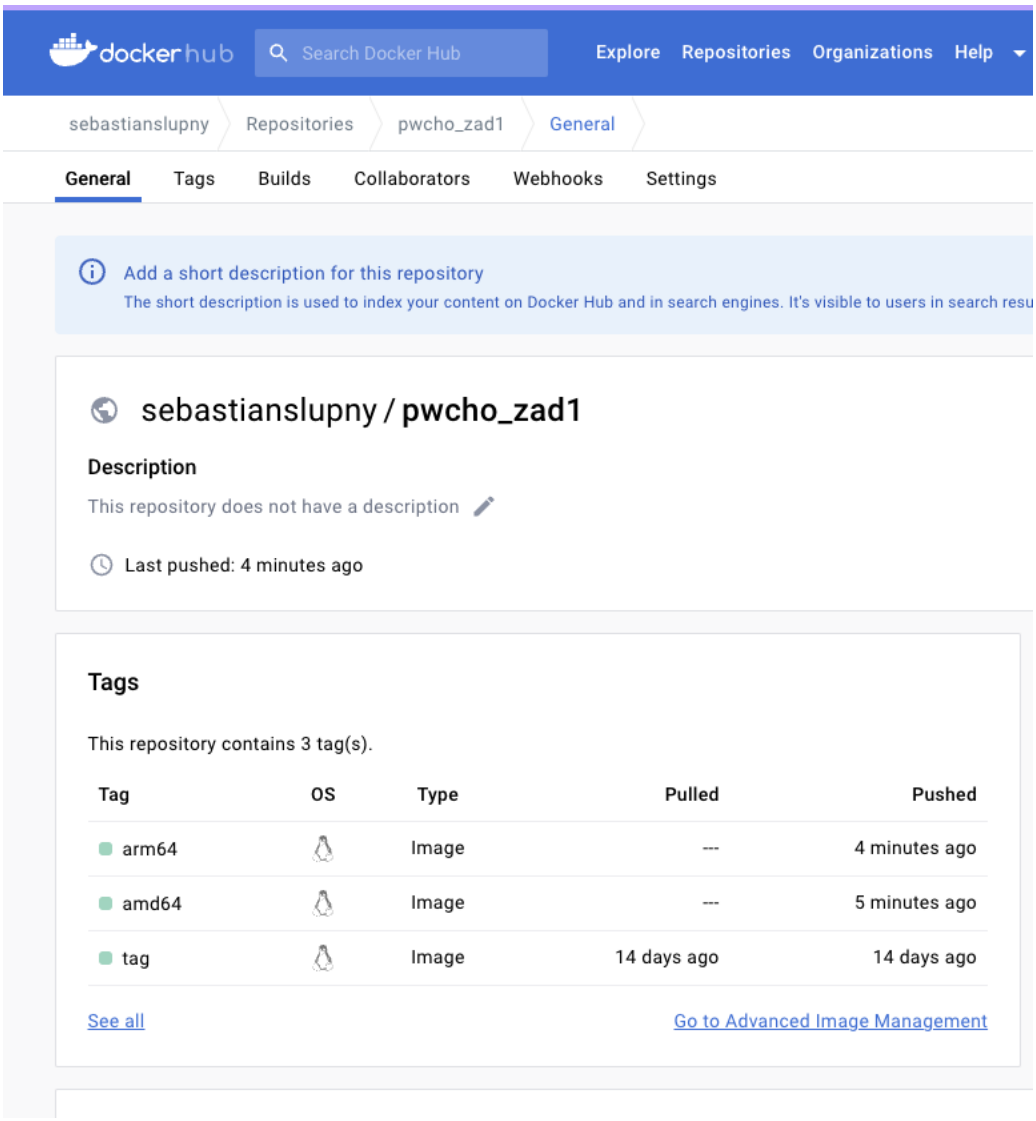
Zrzut stanu build.yaml



stanu build.yaml

Jak możemy zobaczyć Actions nie wykazał problemów przy wykonaniu zadania

Po przejściu na hub.docker.com widzimy ze nasze obrazy są poprawne



Podsumowanie:

Przeprowadziłem proces budowy i publikacji obrazów kontenera za pomocą łańcucha Github Actions. Wykorzystałem plik konfiguracyjny build.yaml, który opisuje wszystkie kroki do wykonania.

Najpierw skonfigurowałem zdarzenie, które wyzwala łańcuch. W moim przypadku zdarzenie to push do gałęzi main. Gdy wprowadziłem zmiany i wrzuciłem je do repozytorium, łańcuch został uruchomiony automatycznie.

W kolejnych krokach skonfigurowałem środowisko. Użyłem akcji actions/checkout@v3, aby pobrać kod aplikacji. Następnie skonfigurowałem QEMU i Docker Buildx za pomocą akcji docker/setup-qemu-action@v2 i docker/setup-buildx-action@v2, aby umożliwić budowę obrazów dla różnych architektur.

Aby móc publikować obrazy, zalogowałem się do Docker Hub przy użyciu akcji docker/login-action@v2. Podając swoje dane uwierzytelniające jako zmienne tajne, byłem w stanie zalogować się bezpiecznie.

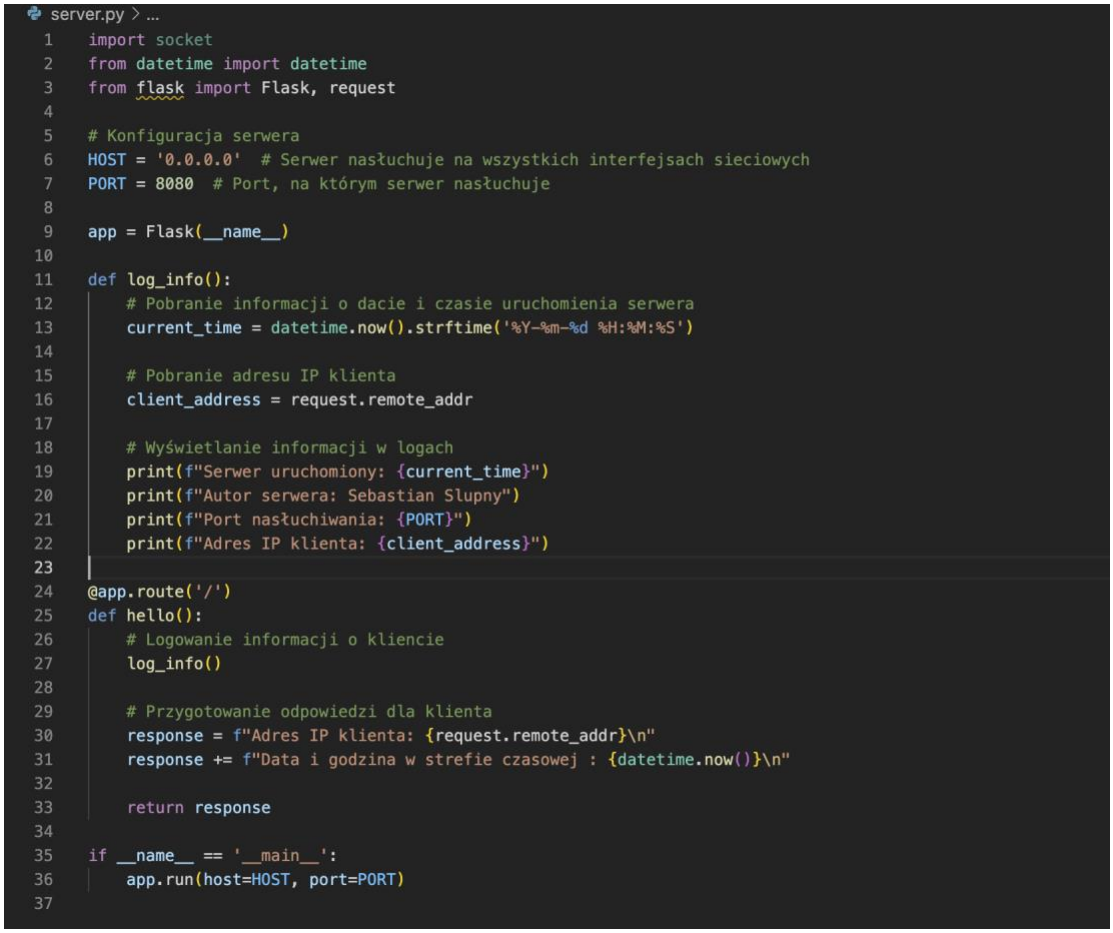
Następnie przyszedł czas na budowanie i publikację obrazów. Użyłem akcji docker/build-push-action@v3, aby skonfigurować proces budowania i publikacji dla architektury linux/amd64. Wskazałem również odpowiednie tagi i platformy, aby obraz został odpowiednio oznaczony i dostępny dla tej architektury.

Analogicznie, użyłem tej samej akcji docker/build-push-action@v3, aby zbudować i opublikować obraz dla architektury linux/arm64/v8. Podobnie jak wcześniej, określiłem tagi i platformy, aby obraz był dostępny dla tej konkretnego architektury.

Po zakończeniu wszystkich kroków, łańcuch Github Actions został wykonany. Sprawdziłem logi, aby upewnić się, że wszystkie kroki zostały wykonane poprawnie, a obrazy Docker zostały zbudowane i opublikowane dla obu architektur.

Teraz mogę potwierdzić, że łańcuch Github Actions został pomyślnie wykonany i obrazy kontenera dla architektur linux/arm64/v8 i linux/amd64 zostały zbudowane i opublikowane. Moja aplikacja jest teraz dostępna i gotowa do użycia na różnych platformach.

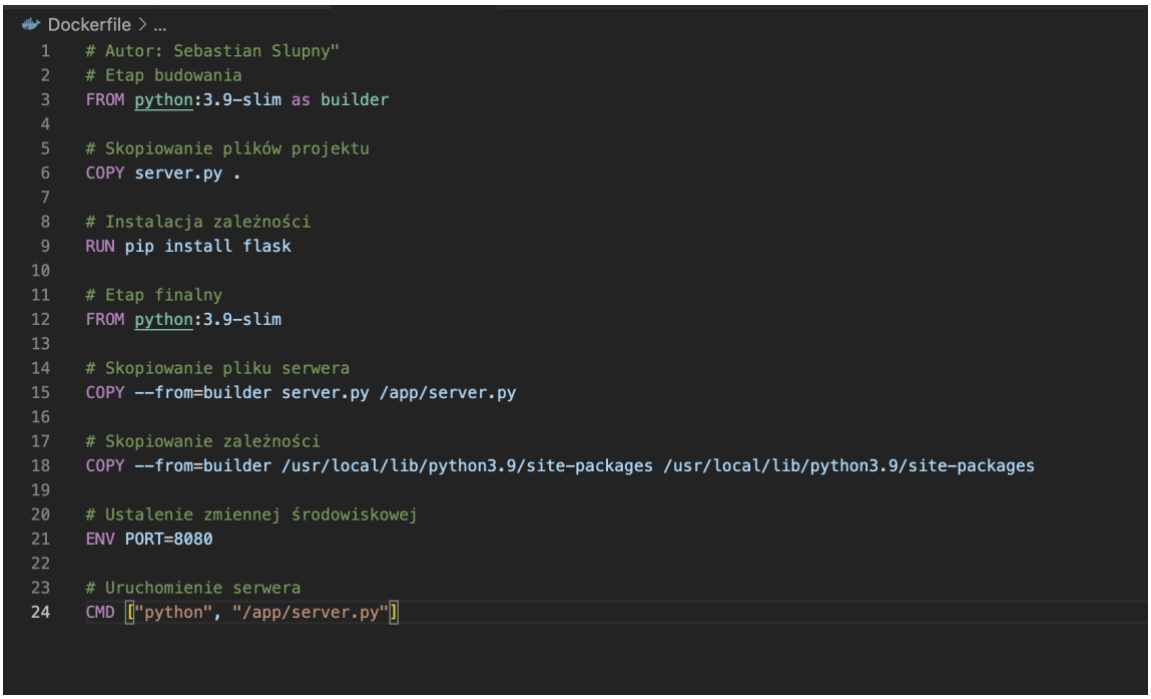
Opracowany plik serwer.py



```python
import socket
from datetime import datetime
from flask import Flask, request

# Konfiguracja serwera
HOST = '0.0.0.0'  # Serwer nasłuchuje na wszystkich interfejsach sieciowych
PORT = 8080  # Port, na którym serwer nasłuchuje

app = Flask(__name__)

def log_info():
    # Pobranie informacji o dacie i czasie uruchomienia serwera
    current_time = datetime.now().strftime('%Y-%m-%d %H:%M:%S')

    # Pobranie adresu IP klienta
    client_address = request.remote_addr

    # Wyświetlanie informacji w logach
    print(f"Serwer uruchomiony: {current_time}")
    print(f"Autor serwera: Sebastian Slupny")
    print(f"Port nasłuchiwania: {PORT}")
    print(f"Adres IP klienta: {client_address}")

@app.route('/')
def hello():
    # Logowanie informacji o kliencie
    log_info()

    # Przygotowanie odpowiedzi dla klienta
    response = f"Adres IP klienta: {request.remote_addr}\n"
    response += f"Data i godzina w strefie czasowej : {datetime.now()}\n"

    return response

if __name__ == '__main__':
    app.run(host=HOST, port=PORT)
```

Kod server.py

Opracowany plik Dockerfile…



```dockerfile
# Autor: Sebastian Slupny"
# Etap budowania
FROM python:3.9-slim as builder

# Skopiowanie plików projektu
COPY server.py .

# Instalacja zależności
RUN pip install flask

# Etap finalny
FROM python:3.9-slim

# Skopiowanie pliku serwera
COPY --from=builder server.py /app/server.py

# Skopiowanie zależności
COPY --from=builder /usr/local/lib/python3.9/site-packages /usr/local/lib/python3.9/site-packages

# Ustalenie zmiennej środowiskowej
ENV PORT=8080

# Uruchomienie serwera
CMD ["python", "/app/server.py"]
```

Plik Dockerfile

Repozytorium GitHub: https://github.com/sebastianslupny4/pwcho_zad1/
Repozytorium DockerHub: https://hub.docker.com/r/sebastianslupny/pwcho_zad1