Cykl życia i narzędzia DevOps - Projekt zaliczeniowy

Autor: Maciej Wiatr 14686

link do repozytorium:

https://github.com/MaciejWiatr/Cykl-zycia-i-narzedzia-DevOps-MWiatr/

Struktura projektu

Projekt sklada się z dwóch katalogów `app/` oraz `.github/`

Katalog app/

Wewnątrz folderu `app/` znajduje się kod źródłowy aplikacji webowej stworzonej przy użyciu technologii Node.js, TypeScript oraz frameworka webowego Hono wraz z testami sprawdzającymi poprawność wyświetlanych danych jak i plikiem `Dockerfile` opisującym proces budowania obrazu dockerowego

Opis Dockerfile

Zawartość pliku Dockerfile

```
FROM node:20-alpine AS base

# Development stage
FROM base AS dev
WORKDIR /app

COPY ./ ./
RUN npm ci

# Build stage
FROM base AS build
RUN apk add --no-cache gcompat
WORKDIR /app

COPY package*.json tsconfig.json ./
COPY src/ ./src/
COPY static/ ./static/
RUN npm ci

RUN npm run build && \
    npm prune --production

# Runner stage
FROM base AS runner
WORKDIR /app

RUN addgroup --system --gid 1001 nodejs
RUN adduser --system --uid 1001 hono

COPY --from=build --chown=hono:nodejs /app/node_modules /app/no
COPY --from=build --chown=hono:nodejs /app/dist /app/dist
COPY --from=build --chown=hono:nodejs /app/package.json /app/pa
COPY --from=build --chown=hono:nodejs /app/static /app/static

USER hono
```

```
EXPOSE 3000

CMD ["node", "/app/dist/index.js"]
```

W Dockerfile definiuje wieloetapowy proces budowania obrazu aplikacji. Składa się z następujących etapów:

Base Stage:

- Bazuje na lekkim obrazie node:20-alpine i służy jako podstawa dla pozostałych etapów.

Development Stage (dev):

- Rozszerza base, kopiuje pliki źródłowe i instaluje wszystkie zależności za pomocą npm ci.

Build Stage (build):

- Rozszerza base, dodaje gcompat, kopiuje niezbędne pliki projektu (package*.json, tsconfig.json, src, static), instaluje zależności, buduje aplikację i usuwa zbędne zależności deweloperskie.

Runner Stage (runner):

- Rozszerza base, tworzy użytkownika systemowego hono i kopiuje artefakty z etapu build. Na koniec ustawia użytkownika hono, konfiguruje port 3000 i definiuje polecenie uruchamiające aplikację.

Katalog .github/

Katalog .github zawiera dwa pliki yaml opisujące worflows github actions

pr-build-text.yml

```
name: PR Build and Test

on:
  pull_request:
    types: [opened, synchronize, reopened]

jobs:
```

```
  build:
    runs-on: ubuntu-latest
    permissions:
      contents: read
      security-events: write
      pull-requests: write

    steps:
    - name: Checkout code
      uses: actions/checkout@v2

    - name: Build Docker image up to dev stage
      run: docker build --target dev -t app:dev -f app/Dockerfi

    - name: Run tests inside Docker
      run: docker run --rm app:dev npm run test:prod

    - name: Build final Docker image
      run: docker build -t app:production -f app/Dockerfile app

    - name: Run Trivy vulnerability scanner
      uses: aquasecurity/trivy-action@master
      with:
        image-ref: 'app:production'
        format: 'sarif'
        output: 'trivy-results.sarif'
        severity: 'CRITICAL,HIGH'
        github-pat: ${{ secrets.GITHUB_TOKEN }}

    - name: Upload Trivy scan results to GitHub Security tab
      uses: github/codeql-action/upload-sarif@v3
      with:
        sarif_file: 'trivy-results.sarif'
```

Workflow `pr-build-text.yml` jest uruchamiany przy każdym otwarciu, synchronizacji lub ponownym otwarciu pull requesta. Wykonuje następujące kroki:

- **Checkout code:** Pobiera kod źródłowy z repozytorium
- **Build Docker image up to dev stage:** Buduje obraz Dockerowy do etapu dev, który zawiera wszystkie zależności deweloperskie
- **Run tests inside Docker:** Uruchamia testy jednostkowe wewnątrz kontenera Docker
- **Build final Docker image:** Buduje finalny obraz produkcyjny aplikacji
- **Run Trivy vulnerability scanner:** Skanuje zbudowany obraz pod kątem podatności bezpieczeństwa, skupiając się na krytycznych (CRITICAL) i wysokich (HIGH) zagrożeniach
- **Upload Trivy scan results:** Wysyła wyniki skanowania do zakładki Security w GitHub, gdzie można je przeglądać i analizować

---

master-deploy.yml

```
name: Build and Deploy

on:
  push:
    branches:
      - master

jobs:
  build-and-deploy:
    runs-on: ubuntu-latest
    permissions:
      contents: read
      packages: write
      security-events: write
      pull-requests: write

    steps:
    - name: Checkout code
      uses: actions/checkout@v2
```

```
    - name: Login to GitHub Container Registry
      uses: docker/login-action@v2
      with:
        registry: ghcr.io
        username: ${{ github.actor }}
        password: ${{ secrets.GHCR_TOKEN }}

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

    - name: Build Docker image up to dev stage
      run: docker build --target dev -t app:dev -f app/Dockerfi

    - name: Run tests inside Docker
      run: docker run --rm app:dev npm run test:prod

    - name: Build and push final Docker image
      uses: docker/build-push-action@v4
      with:
        context: ./app
        file: ./app/Dockerfile
        push: false
        load: true
        tags: |
          app:production
          ghcr.io/maciejwiatr/cykl-zycia-i-narzedzia-devops:late
          ghcr.io/maciejwiatr/cykl-zycia-i-narzedzia-devops:${{

    - name: Run Trivy vulnerability scanner
      uses: aquasecurity/trivy-action@master
      with:
        image-ref: 'app:production'
        format: 'sarif'
        output: 'trivy-results.sarif'
        severity: 'CRITICAL,HIGH'
        github-pat: ${{ secrets.GITHUB_TOKEN }}
```

```
    - name: Upload Trivy scan results to GitHub Security tab
      uses: github/codeql-action/upload-sarif@v3
      with:
        sarif_file: 'trivy-results.sarif'

    - name: Push Docker image to GitHub registry
      run: |
        docker push ghcr.io/maciejwiatr/cykl-zycia-i-narzedzia-d
        docker push ghcr.io/maciejwiatr/cykl-zycia-i-narzedzia-d
```

Workflow `master-deploy.yml` jest uruchamiany przy każdym pushu do brancha master. Wykonuje następujące kroki:

- **Checkout code:** Pobiera kod źródłowy z repozytorium
- **Login to GitHub Container Registry:** Loguje się do rejestru kontenerów GitHub używając tokenu z repository secrets
- **Set up Docker Buildx:** Konfiguruje Docker Buildx do budowania obrazów
- **Build Docker image up to dev stage:** Buduje obraz deweloperski z wszystkimi zależnościami
- **Run tests inside Docker:** Uruchamia testy jednostkowe w kontenerze
- **Build and push final Docker image:** Buduje finalny obraz produkcyjny i taguje go odpowiednimi wersjami (latest oraz SHA commita)
- **Run Trivy vulnerability scanner:** Skanuje zbudowany obraz pod kątem podatności bezpieczeństwa
- **Upload Trivy scan results:** Przesyła wyniki skanowania do zakładki Security w GitHub
- **Push Docker image:** Wysyła zbudowany obraz do GitHub Container Registry z dwoma tagami - latest oraz SHA commita

Workflow wykorzystuje sekret `GHCR_TOKEN` do uwierzytelnienia w rejestrze kontenerów aby przesłać obraz dockerowy do github container registry

Instrukcja uruchomienia aplikacji

Aby uruchomić aplikację lokalnie należy wykonać następujące kroki:

Wymagania

- Docker zainstalowany na systemie
- Git zainstalowany na systemie

Kroki

1. Sklonuj repozytorium:

```
git clone https://github.com/MaciejWiatr/Cykl-zycia-i-narzedz
cd cykl-zycia-i-narzedzia-devops
```

2. Przejdź do katalogu aplikacji:

```
cd app
```

3. Zbuduj obraz dockerowy:

```
docker build -t app:latest .
```

4. Uruchom kontener:

```
docker run -p 3000:3000 app:latest
```

Aplikacja będzie dostępna pod adresem `http://localhost:3000`

Uruchamianie w trybie deweloperskim

Aby uruchomić aplikację w trybie deweloperskim, który umożliwia hot-reloading:

1. Zbuduj obraz deweloperski:

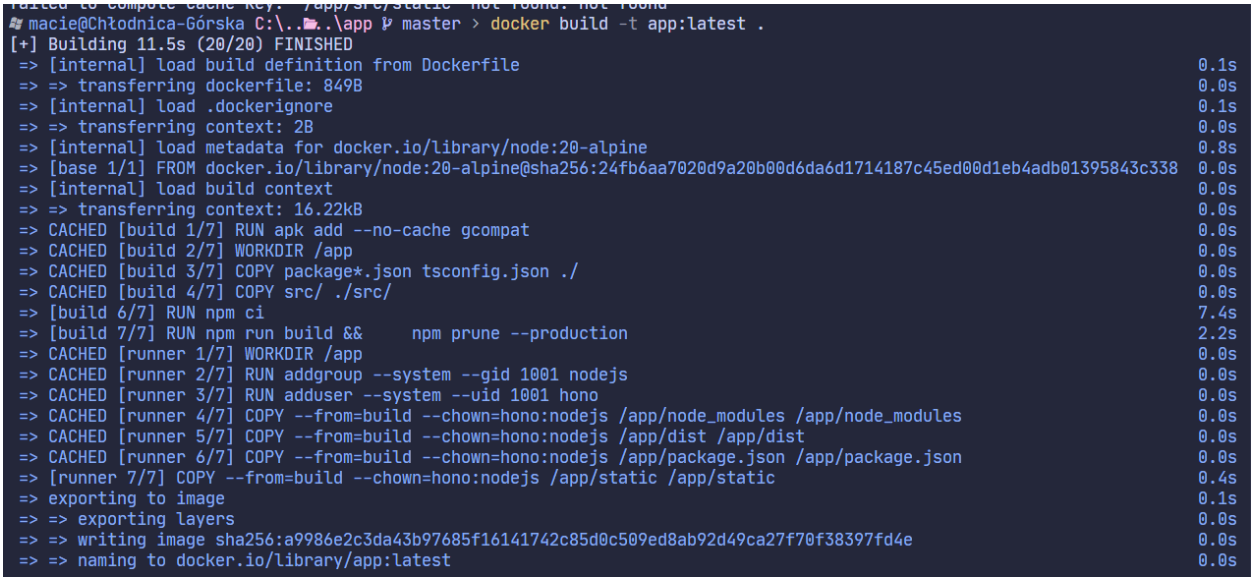
```
docker build --target dev -t app:dev .
```

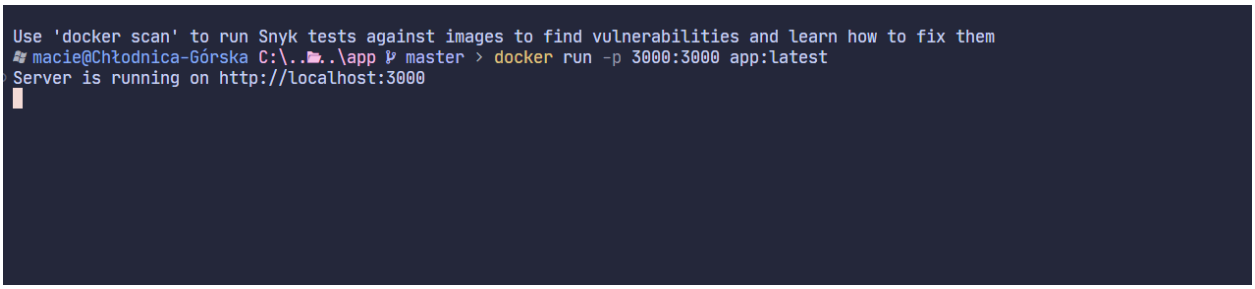
2. Uruchom kontener w trybie deweloperskim:

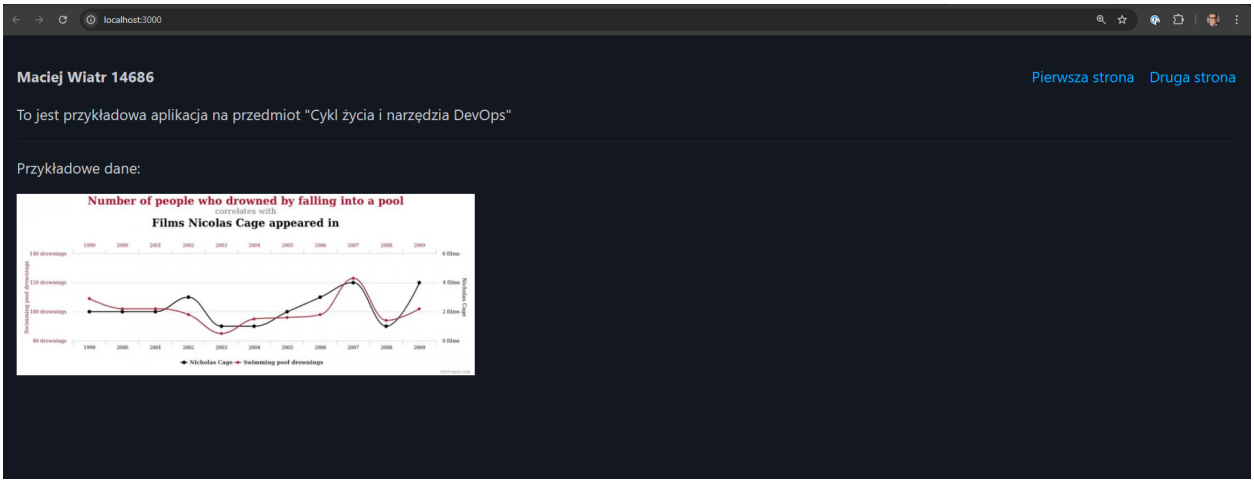
```
docker run -p 3000:3000 -v $(pwd)/src:/app/src app:dev npm ru
```

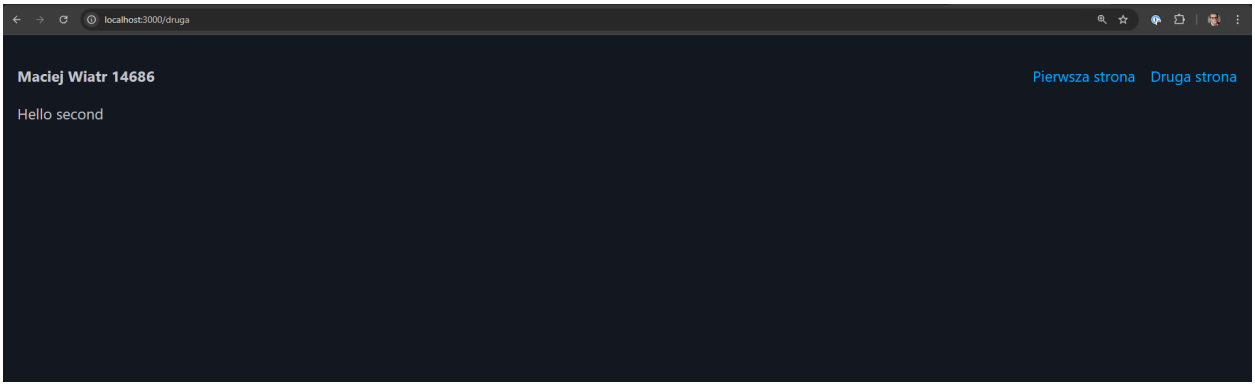
Aplikacja w trybie deweloperskim będzie dostępna pod tym samym adresem `http://localhost:3000` i będzie automatycznie przeładowywać się przy zmianach w kodzie.

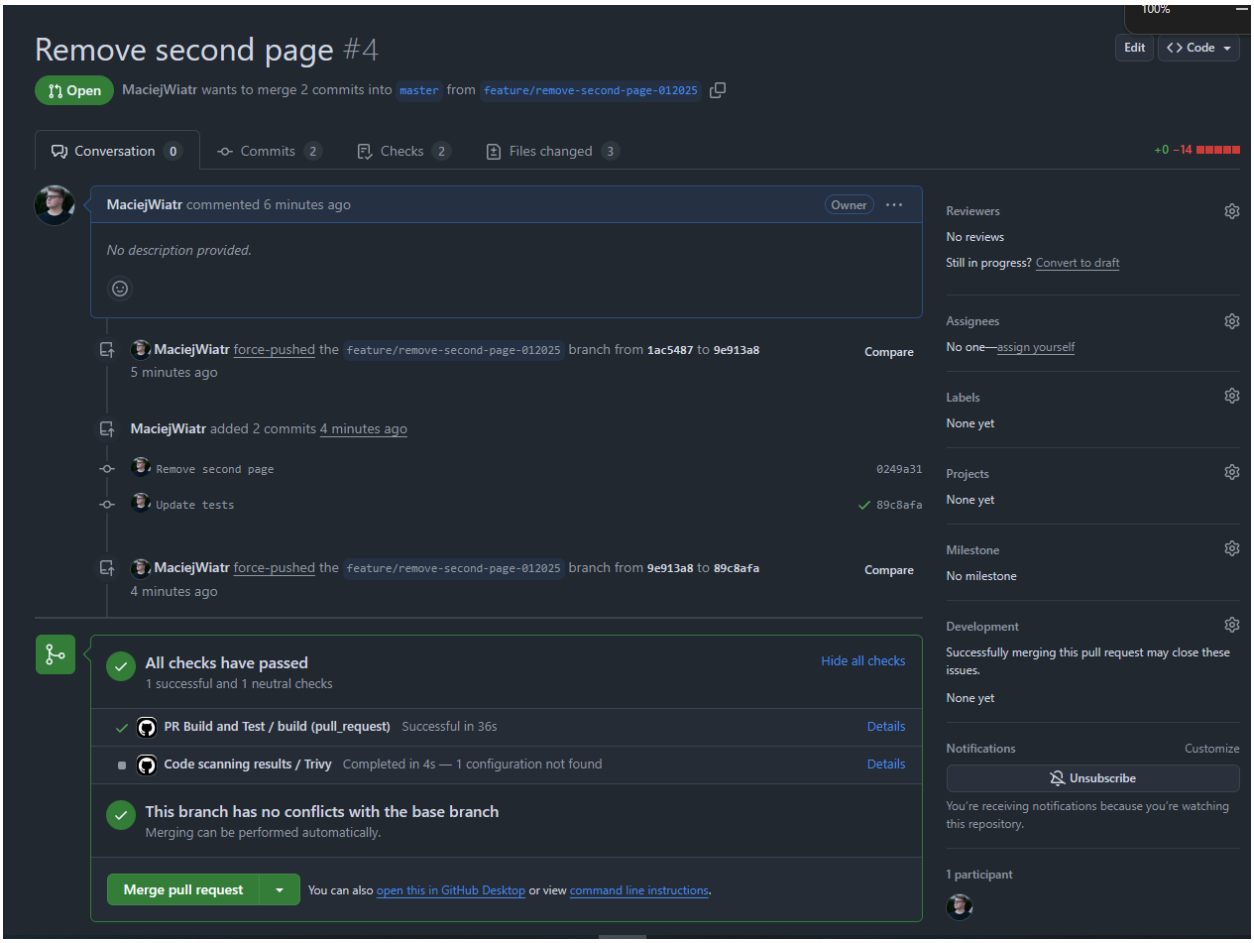
Screenshoty uruchomionej aplikacji






localhost:3000
Maciej Wiatr 14686
Pierwsza strona
Druga strona
To jest przykładowa aplikacja na przedmiot "Cykl życia i narzędzia DevOps"
Przykładowe dane:
Number of people who drowned by falling into a pool
correlates with
Films Nicolas Cage appeared in
Nicholas Cage
Swimming pool drownings


localhost:3000/druga
Maciej Wiatr 14686
Pierwsza strona
Druga strona
Hello second


Remove second page #4
Edit
Code
Open
MaciejWiatr wants to merge 2 commits into master from feature/remove-second-page-012025
Conversation 0
Commits 2
Checks 2
Files changed 3
+0 −14
MaciejWiatr commented 6 minutes ago
Owner
No description provided.
MaciejWiatr force-pushed the feature/remove-second-page-012025 branch from 1ac5487 to 9e913a8
5 minutes ago
Compare
MaciejWiatr added 2 commits 4 minutes ago
Remove second page
0249a31
Update tests
89c8afa
MaciejWiatr force-pushed the feature/remove-second-page-012025 branch from 9e913a8 to 89c8afa
4 minutes ago
Compare
All checks have passed
1 successful and 1 neutral checks
Hide all checks
PR Build and Test / build (pull_request) Successful in 36s
Details
Code scanning results / Trivy Completed in 4s — 1 configuration not found
Details
This branch has no conflicts with the base branch
Merging can be performed automatically.
Merge pull request
You can also open this in GitHub Desktop or view command line instructions.
Reviewers
No reviews
Still in progress? Convert to draft
Assignees
No one—assign yourself
Labels
None yet
Projects
None yet
Milestone
No milestone
Development
Successfully merging this pull request may close these issues.
None yet
Notifications
Customize
Unsubscribe
You're receiving notifications because you're watching this repository.
1 participant