



Docker

Docker to platforma open-source do tworzenia, pakowania i uruchamiania aplikacji w kontenerach. Kontener to lekka izolowana paczka zawierająca:

- kod aplikacji.
- wszystkie potrzebne biblioteki i narzędzia uruchomieniowe (np. JRE, Python).
- Minimalną część systemu potrzebną do działania aplikacji.



Obraz (Image)

- Niezmienny szablon, Można go porównać do migawki systemu plików. Obrazy składają się z warstw, co pozwala na efektywne zarządzanie przestrzenią dyskową i szybkie budowanie nowych obrazów na podstawie już istniejących. Obrazy mogą zawierać system operacyjny (lub jego minimalną wersję), środowisko uruchomieniowe (np. Java), biblioteki oraz kod aplikacji.

Kontener (Container)

- Działająca instancja obrazu Docker. Jest to lekkie, izolowane środowisko, które uruchamia aplikację. Każdy kontener posiada własny system plików, przestrzeń procesów i interfejs sieciowy, ale współdzieli jądro systemu operacyjnego hosta z innymi kontenerami. Dzięki temu kontenery uruchamiają się znacznie szybciej i zużywają mniej zasobów niż tradycyjne maszyny wirtualne.



Dockerfile

- Plik tekstowy zawierający sekwencję poleceń (instrukcji), które Docker wykonuje w celu automatycznego zbudowania obrazu Docker. Definiuje on m.in. obraz bazowy, kopiowane pliki, instalowane zależności, ustawiane zmienne środowiskowe oraz komendę uruchamiającą aplikację w kontenerze.

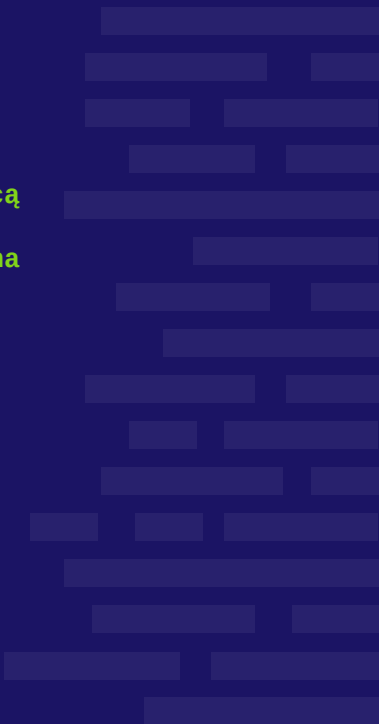
Rejestr (Registry)

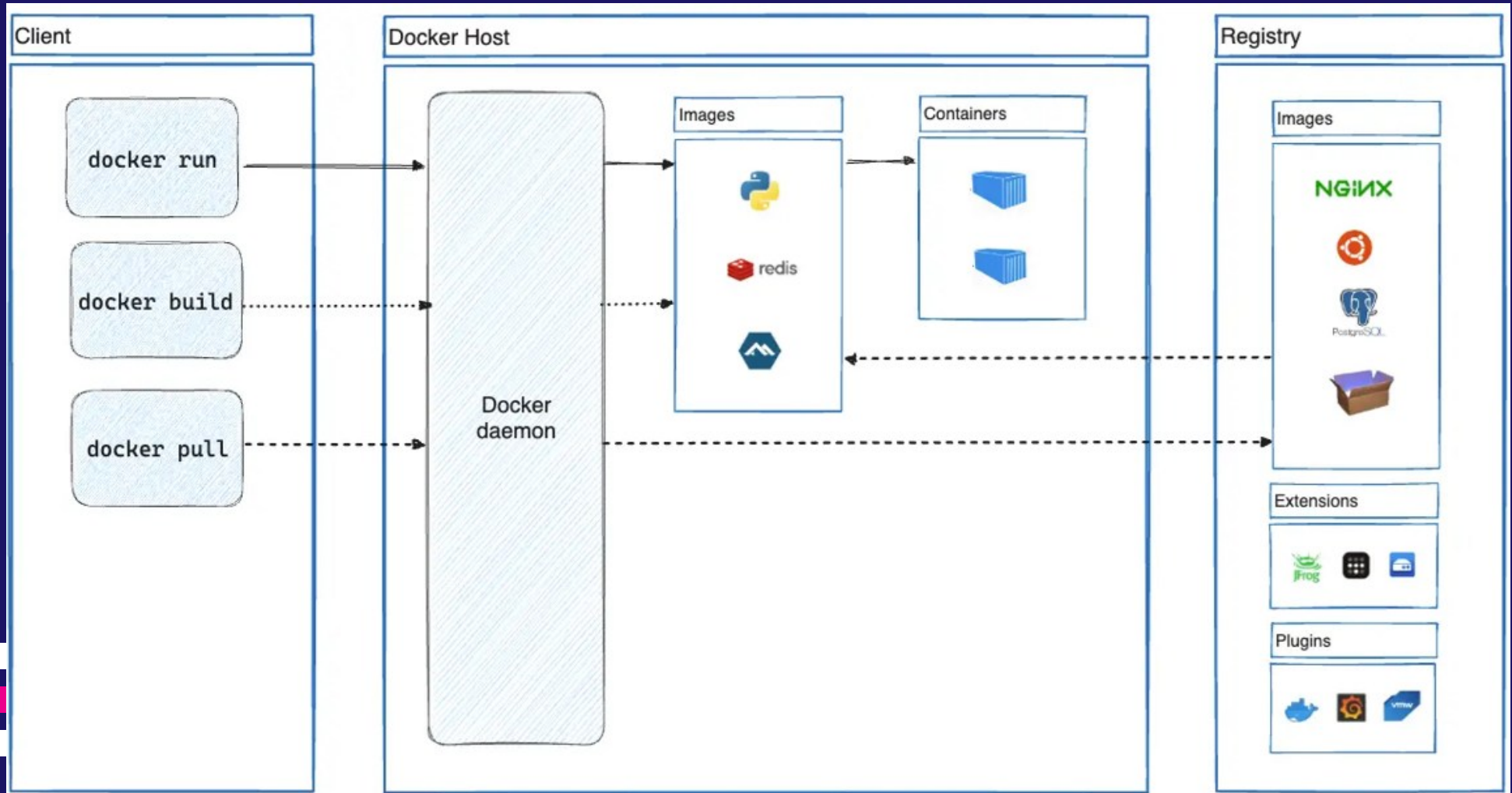
- System służący do przechowywania i dystrybucji obrazów Docker. Najbardziej znanym publicznym rejestrem jest Docker Hub, gdzie można znaleźć tysiące gotowych obrazów dla popularnych technologii i aplikacji. Możliwe jest również tworzenie prywatnych rejestrów do przechowywania własnych, firmowych obrazów.



Silnik (Engine)

- Podstawowa technologia typu klient-serwer, która tworzy i uruchamia kontenery Docker. Składa się z demona Docker (procesu działającego w tle), interfejsu API REST, za pomocą którego programy mogą komunikować się z demonem, oraz klienta wiersza poleceń (CLI), który pozwala użytkownikom na interakcję z Dockerem.







Element	Przepływ
1 Client	terminal lub GUI (Docker Desktop). Wysyłasz komendy: <ul style="list-style-type: none">• docker build – buduj obraz z Dockerfile• docker run – uruchom kontener z obrazu• docker pull – pobierz obraz z rejestru
2 Docker Host	Maszyna, na której działa Docker daemon (dockerd). Demon <ul style="list-style-type: none">— odbiera polecenia klienta,— zarządza lokalnym magazynem Images (szablony) i Containers (uruchomione egzemplarze),— komunikuje się z zewnętrznym rejestrem (pobiera/wysyła warstwy).
3 Registry	Zewnętrzny magazyn obrazów (Docker Hub, GHCR, ECR...). Przechowuje wersje gotowych środowisk: np. Nginx , Ubuntu , PostgreSQL . Może też udostępniać rozszerzenia (Extensions) do zarządzania oraz plug-ins sieci/ storage.

Instalacja

Aby zainstalować Dockera należy:

- posiadać włączoną wirtualizację.

- zainstalować WSL2 (Windows Subsystem for Linux 2)

*Uwaga: należy zrestartować system aby zainstalować ubuntu.

Wykorzystanie	Szybkość	Szybkość podstawowa:	2,60 GHz
4%	1,12 GHz	Gniazda:	1
		Rdzenie:	14
Procesy	Wątki	Dojścia	Procesory logiczne: 20
394	7640	3066704	Wirtualizacja: Włączone
Czas pracy			Pamięć podręczna poziomu 1: 1,2 MB
5:03:04:40			Pamięć podręczna poziomu 2: 11,5 MB
			Pamięć podręczna poziomu 3: 24,0 MB

Administrator: Windows PowerShell

Windows PowerShell

Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! <https://aka.ms/PSWindows>

PS C:\WINDOWS\system32> wsl --install

Downloading: Windows Subsystem for Linux 2.5.7

Installing: Windows Subsystem for Linux 2.5.7

Instalacja

Po restarcie systemu:

- ustawiamy WSL2
- instalujemy Ubuntu
- ustawiamy username i password
- Robimy aktualizacje systemu Ubuntu:
`sudo apt update && sudo apt upgrade -y`

```
PS C:\WINDOWS\system32> wsl --set-default-version 2
Aby uzyskać informacje na temat kluczowych różnic w podsystemie WSL 2, odwiedź stronę https://aka.ms/wsl2
Operacja ukończona pomyślnie.
PS C:\WINDOWS\system32> wsl --install
Pobieranie: Ubuntu
Instalowanie: Ubuntu
Dystrybucja została pomyślnie zainstalowana. Można ją uruchomić za pomocą polecenia "wsl.exe -d Ubuntu"
Trwa uruchamianie Ubuntu...
Provisioning the new WSL instance Ubuntu
This might take a while...
Create a default Unix user account: luke_
```


Instalacja

Pobieramy i instalujemy Docker Desktop:
<https://www.docker.com/products/docker-desktop>

Installing Docker Desktop 4.41.2 (191736)

Docker Desktop 4.41.2

Installation succeeded

You must restart Windows to complete installation.

Close and restart

docker.desktop PERSONAL

Search Ctrl+K ? 2

Settings Give feedback

General

Resources

Docker Engine

Builders

Kubernetes

Software updates

Extensions

Features in development

☐ Expose daemon on tcp://localhost:2375 without TLS

Exposing daemon on TCP without TLS helps legacy clients connect to the daemon. It also makes yourself vulnerable to remote code execution attacks. Use with caution.

☒ Use the WSL 2 based engine (Windows Home can only run the WSL 2 backend)

☐ Add the *.docker.internal names to the host's /etc/hosts file (requires password)

Lets you resolve *.docker.internal DNS names from both the host and your containers. [Learn more](#)

☒ Use containerd for pulling and storing images

The containerd image store enables native support for multi-platform images, attestations, Wasm, and more.

☒ Send usage statistics

Send error reports, system version and language as well as Docker Desktop lifecycle information (e.g., starts, stops, resets).

☐ Use Enhanced Container Isolation

Enhance security by preventing containers from breaching the Linux VM. [Learn more](#)

Business subscription required. [Upgrade](#)

Cancel

Apply & restart



Engine running

RAM 3.14 GB CPU 0.25% Disk: 1.14 GB used (limit 1006.85 GB)

> Terminal Update

Prosta aplikacja

Zbudujemy prostą aplikację w Spring Boot, wykorzystującą połączenie z bazą neon.tech



Project
☐ Gradle - Groovy
☐ Gradle - Kotlin ☒ **Maven**

Language
☒ **Java** ☐ Kotlin ☐ Groovy

Spring Boot
☐ 4.0.0 (SNAPSHOT) ☐ 3.5.1 (SNAPSHOT) ☒ **3.5.0**
☐ 3.4.7 (SNAPSHOT) ☐ 3.4.6 ☐ 3.3.13 (SNAPSHOT) ☐ 3.3.12

Project Metadata

Group

Artifact

Name

Description

Package name

Packaging ☒ **Jar** ☐ War


Java ☐ 24 ☐ 21 ☒ **17**

Dependencies ADD DEPENDENCIES... CTRL + B

Spring Data JPA SQL
Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.

Spring Web WEB
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

PostgreSQL Driver SQL
A JDBC and R2DBC driver that allows Java programs to connect to a PostgreSQL database using standard, database independent Java code.



GENERATE CTRL + G

EXPLORE CTRL + SPACE

...

Prosta aplikacja

Zbudujemy prostą aplikację w Spring Boot, wykorzystującą połączenie z bazą neon.tech

```
@Entity
@Table(name = "products")
public class Product {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;

    public Product() {}
    public Product(String name) { this.name = name; }

    public Long getId() { return id; }
    public void setId(Long id) { this.id = id; }
    public String getName() { return name; }
    public void setName(String name) { this.name = name; }
}
```

```
@Repository
public interface ProductRepository
    extends JpaRepository<Product, Long>
{
}
```

Prosta aplikacja

Zbudujemy prostą aplikację w Spring Boot, wykorzystującą połączenie z bazą neon.tech

```
@RestController
@RequestMapping("/api/products")
public class ProductController {
    private final ProductRepository productRepository;

    @Autowired
    public ProductController
    (ProductRepository productRepository) {
        this.productRepository = productRepository;
    }

    @GetMapping("/{id}")
    public ResponseEntity<Product>
    getProductById(@PathVariable("id") Long id) {
        Optional<Product> productData =
            productRepository.findById(id);
        return productData.map(
            i ->
                new ResponseEntity<>(i, HttpStatus.OK))
            .orElseGet(() ->
                new ResponseEntity<>(HttpStatus.NOT_FOUND));
    }
}
```

```
@PostMapping
public ResponseEntity<Product> createProduct(@RequestBody Product product) {
    try {
        Product savedProduct =
            productRepository.save(new Product(product.getName()));
        return new ResponseEntity<>(savedProduct, HttpStatus.CREATED);
    } catch (Exception e) {
        return new ResponseEntity<>(null, HttpStatus.INTERNAL_SERVER_ERROR);
    }
}

@GetMapping
public ResponseEntity<List<Product>> getAllProducts() {
    try {
        List<Product> products = productRepository.findAll();
        if (products.isEmpty()) {
            return new ResponseEntity<>(HttpStatus.NO_CONTENT);
        }
        return new ResponseEntity<>(products, HttpStatus.OK);
    } catch (Exception e) {
        return new ResponseEntity<>(null, HttpStatus.INTERNAL_SERVER_ERROR);
    }
}
```

Prosta aplikacja

application.properties

```
spring.application.name=demo

server.port=8080

spring.datasource.url=${DB_CONNECT_URL}

spring.datasource.username=${DB_USER}

spring.datasource.password=${DB_PASSWORD}

spring.datasource.driver-class-name=org.postgresql.Driver

spring.jpa.database-platform=org.hibernate.dialect.PostgreSQLDialect

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=true

spring.jpa.properties.hibernate.format_sql=true
```

Dockerfile

W katalogu głównym projektu tworzymy plik Dockerfile. Dockerfile to deklaracyjny plik opisujący, jak zbudować finalny artefakt — obraz kontenera z aplikacją i wszystkimi zależnościami.

Od Docker 17.05 można umieścić w jednym Dockerfile kilka sekcji FROM - Każda tworzy oddzielny etap budowy.

Dockerfile

builder	Pełne środowisko deweloperskie (np. Maven). Kompiluje, testuje, generuje artefakt (jar, statyczne pliki).	Narzędzia potrzebne tylko do kompilacji.
runtime	Minimalna zawartość do uruchomienia aplikacji (np. Alpine JRE). Kopiuje tylko gotowy artefakt z etapu builder.	Finalny obraz - mały, szybki, pozbawiony niepotrzebnych narzędzi.

Dockerfile

Etapy w Dockerfile:

Builder:

```
# ----- Etap 1: Builder -----  
FROM maven:3.9-eclipse-temurin-17 AS builder  
WORKDIR /app  
COPY pom.xml .  
RUN mvn -B dependency:go-offline  
COPY src ./src  
RUN mvn -B package -DskipTests  
# ----- Etap 2: runtime -----  
FROM eclipse-temurin:17-jre-alpine  
ENV DB_CONNECT_URL="" \  
    DB_USER="" \  
    DB_PASSWORD="" \  
    JAVA_OPTS=""  
WORKDIR /app  
COPY --from=builder /app/target/*.jar app.jar  
EXPOSE 8080  
ENTRYPOINT ["sh", "-c", "java $JAVA_OPTS -jar /app/app.jar"]
```

Zaczynamy od obraz buildera (maven z jdk temurin na ubuntu)
Ustawiamy katalog roboczy
Kopiujemy do niego plik z zależnościami
Pobieramy zależności raz, nie trzeba w kolejnych etapach
Kopiujemy kod projektu
Budujemy za pomocą mavena (pomijamy testy)

Runtime:

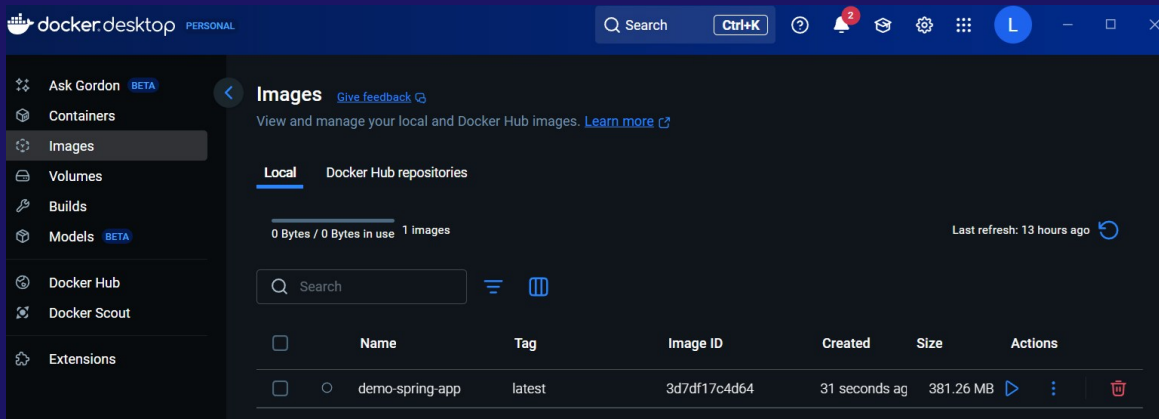
Zaczynamy od obrazu z jre temurin na alpine
wpisujemy zmienne środowiskowe (puste)
ustawiamy katalog roboczy
Kopiujemy artefakt z poprzedniego etapu do app.jar
ustawiamy port wewnątrz dockera na 8080
Definiujemy główny proces w kontenerze – uruchamiane przy starcie.

Każda instrukcja w Dockerfile generuje migawkę systemu plików jedną warstwę. Warstwy są niezmiennie. Docker przechowuje wykonane warstwy w cache. Przy kolejnym buildzie/runie sprawdza, czy treść instrukcji i plików się nie zmieniła. Jeżeli wszystko pasuje, używa warstwy z cache zamiast wykonywać krok jeszcze raz (oszczędza czas i transfer). Finalny obraz to złożenie wszystkich warstw jeden na drugim; przy uruchomieniu kontenera Docker dokłada jeszcze warstwę „read-write”, w której działającej aplikacji wolno zapisywać pliki.

Dockerfile

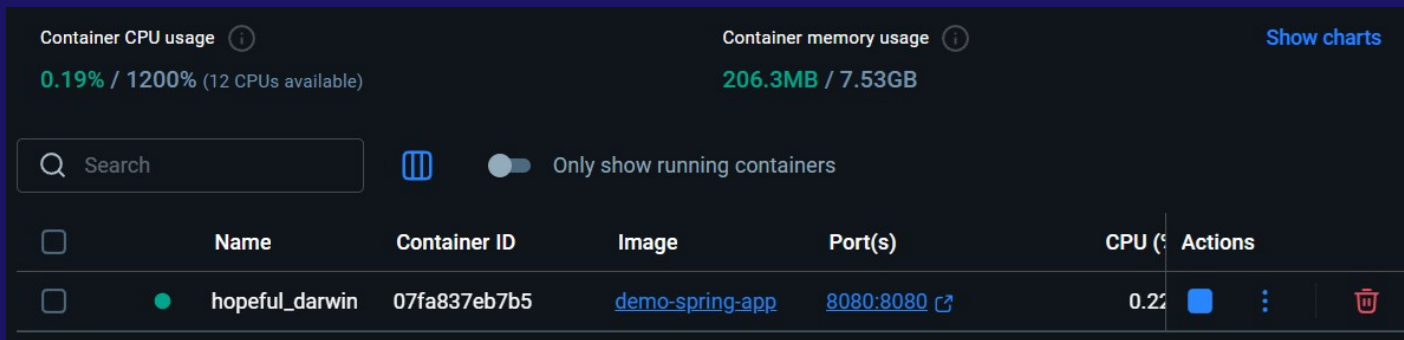
Budujemy obraz

```
docker build -t demo-spring-app .
```



Uruchamiamy kontener

```
docker run --rm -p 8080:8080 -e DB_CONNECT_URL="jdbc:postgresql://abc.neon.tech/neondb?sslmode=require" -e DB_USER="neondb_owner" -e DB_PASSWORD="abc" demo-spring-app
```



Render

Pamiętamy o zmiennych środowiskowych,
Render będzie działał na innym porcie niż
lokalnie!

Zakładamy konto za pomocą
github'a na render.com

Tworzymy nowy WebService

Key	Value
DB_CONNECT_URL
DB_PASSWORD
DB_USER
SERVER_PORT	10000

Configure and deploy your new Web Service

✓ Choose service > 2 Configure > 3 Deploy

Need help? Docs ↗

Source Code

Git Provider Public Git Repository Existing Image

PR Previews and Auto-Deploy are available only for repositories configured with render.yaml

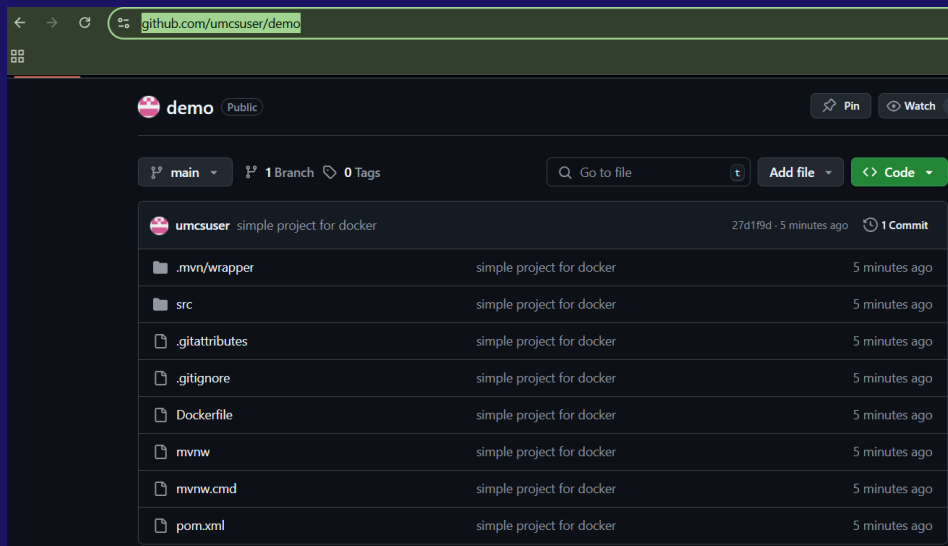
🌐 <https://github.com/umcsuser/demo>

Connect →

Render

Dodajemy do repozytorium
github:

```
git init
git add .
git commit -m "first commit"
git branch -M main
git remote add origin
https://github.com/umcsuser/demo.git
git push -u origin main
```



Render

Pamiętamy o zmiennych środowiskowych,
Render będzie działał na innym porcie niż
lokalnie!

Zakładamy konto za pomocą
github'a na render.com

Tworzymy nowy WebService

Key	Value
DB_CONNECT_URL
DB_PASSWORD
DB_USER
SERVER_PORT	10000

Configure and deploy your new Web Service

✓ Choose service > 2 Configure > 3 Deploy

Need help? Docs ↗

Source Code

Git Provider Public Git Repository Existing Image

PR Previews and Auto-Deploy are available only for repositories configured with render.yaml

🌐 <https://github.com/umcsuser/demo>

Connect →

Render

Ungrouped Services

Active (1)

Suspended (0)

All (1)

🔍 Search services

SERVICE NAME ¹	STATUS	RUNTIME	REGION	DEPLOYED ↓	
🌐 <u>demo</u>	✓ Deployed	Docker	Frankfurt	57min	...

Postman

The screenshot displays the Postman application interface. At the top, a POST request is configured for the URL `http://localhost:8080/api/products`. The **Body** tab is selected, showing a JSON payload: `{ "name": "karma dla psa" }`. Below the request configuration, the **Body** tab of the response section is active, displaying the received JSON: `{ "id": 2, "name": "karma dla psa" }`. The status bar at the bottom indicates a successful response with status 201 (Created), a time of 1066 ms, and a size of 200 B.

Request Configuration:

- Method: POST
- URL: `http://localhost:8080/api/products`
- Body Type: JSON
- Body Content:

```
{
  "name": "karma dla psa"
}
```

Response Details:

- Status: 201 Created
- Time: 1066 ms
- Size: 200 B
- Body Content:

```
{
  "id": 2,
  "name": "karma dla psa"
}
```

Postman

GET

https://demo-qa94.onrender.com/api/products

Send

Params

Authorization

Headers (6)

Body

Pre-request Script

Tests

Settings

Cookies

Query Params

	Key	Value	Bulk Edit
	Key	Value	

Body

Cookies

Headers (12)

Test Results

Status: 200 OK Time: 2 m 8.28 s Size: 418 B Save Response

Pretty

Raw

Preview

Visualize

JSON

```
1  [
2    {
3      "id": 1,
4      "name": "karma dla kota"
5    },
6    {
7      "id": 2,
8      "name": "karma dla psa"
9    }
10 ]
```

Dziękuję za uwagę!

CREDITS: This presentation template was created by Slidesgo, including icons by Flaticon, and infographics & images by Freepik.