




**Uniwersytet Gdański**
**Wydział Matematyki, Fizyki i Informatyki**
**Instytut Informatyki**

# Movies Service

**Wiktor Szymulewicz**

Projekt z przedmiotu technologie chmurowe na kierunku informatyka profil praktyczny na Uniwersytecie Gdańskim.

**Gdańsk**

**25 czerwca 2024**

# Spis treści

# 1 Opis projektu

Aplikacja ta powstała z potrzeby o bardzo wydajną aplikację, w której użytkownicy mogą przeglądać filmy, oraz zostawiać swoje recenzje. Movies service pozwala adminom na dodawanie nowych filmów, z opcją publikacja zdjęć, a użytkownicy mogą przeglądać filmy, dodawać do swojej watchlisty, do ulubionych, zostawiać recenzje, oraz komentarze. Aplikacja zapewnia również powiadmienia w czasie rzeczywistym za pomocą protokuły MQTT, takie jak notyfikacje, oraz publiczny czat.

## 1.1 Opis architektury

Aplikacja opiera się na mikroserwisach, działających w kontenerach Docker, zarządzanych przez Kubernetes. Do mikroserwisów, należy grafowa baza danych Neo4j, REST API w ASP.NET Core, i interfejs użytkownika NEXT.js. Komunikacja między mikroserwisami odbywa się przez protokoły HTTP i bolt.

- **Grafowa Baza Danych Neo4j** – Wykorzystywana do przechowywania i zarządzania danymi w formie grafowej.
- **REST API w ASP.NET Core** – Pozwala na komunikację z bazą danych, otrzymuje requesty HTTP, oraz nasłuchuje tematy MQTT.
- **Interfejs Użytkownika NEXT.js** – Aplikacja frontendowa obsługująca interakcje użytkownika.

Architektura aplikacji w Kubernetes obejmuje następujące komponenty:

- **Deploymenty/StatefulSety** – Definiują sposób wdrożenia każdego mikroserwisu. W tej aplikacji, deploymenty głównie konfigurują kontenery.
- **Serwisy (Services)** – Umożliwiają na komunikajce między mikroserwisami. Każdy serwis w tej aplikacji jest typu ClusterIP, co oznacza, dostęp do serwisu jest możliwy tylko wewnątrz klastra.
- **Persistent Volumes (PV)** – Długotrwałe zasoby dyskowe. Umożliwiają zachowanie danych, nawet kiedy pod bazy danych jest restartowany.
- **Persistent Volume Claims (PVC)** – Umożliwia bazie danych żądanie do zasobów PV.
- **Sekrety (Secrets)** – Bezpieczne przechowywanie tajnych informacji, takie jak klucze API do zewnętrznych zasobów (takich jak Cloudinary), oraz hasła.
- **Konfiguracje (ConfigMaps)** – Zawierują konfiguracje, głównie zmienne środowiskowe oraz mapowanie wolumenów.
- **Ingressy** – Zarządzają ruchem sieciowym. Umożliwiają na zewnętrzny dostęp do serwisów typu ClusterIP, poprzez przekierowywanie rządań.

## 1.2 Opis infrastruktury

- **Narzędzia** – Aplikacja będzie postawiona w AKS (Azure Kubernetes Service). Narzędzie helm umożwliwi instalowanie pakietów.
- **Dostęp** – Dostęp do aplikacja będzie możliwy za pomocą TLS. Użytkownik będzie miał jedyniu dostęp do UI oraz API.
- **Monitorowanie** – Monitorowanie będzie możliwe za pomocą Azure Portal.

## 1.3 Opis komponentów aplikacji

## 1.4 Wewnętrzne komponenty aplikacji

### 1.4.1 Neo4j (Baza danych grafowa)

- **Opis**: Neo4j jest graficzną bazą danych, co jest doskonałym modelem dla danych używanych przez Movies Service. Dodana jest również paczka APOC, co umożliwia między innymi na wykonywanie zaplanowanych zadań.
- **Wdrażanie**: Neo4j jest wdrażany z oficjalnego obrazu jako StatefulSet, co gwarantuje trwałość danych oraz bardziej przewidywalne nazwy podów. PVC jest używany do przechowywania trwałych danych, dzięki czemu dane są przechowywane nawet po zatrzymywaniach replik.
- **Konfiguracja**: Konfiguracja Neo4j odbywa się za pomocą ConfigMaps oraz Secrets w Kubernetes, gdzie przechowywane są ustawienia konfiguracyjne oraz dane uwierzytelniające.

### 1.4.2 ASP.NET (Backend)

- **Opis**: ASP.NET Core jest używany głównie do tworzenia aplikacji backendowych. Framework ten umożliwia na tworzenie bardzo wydajnych API.
- **Wdrażanie**: Aplikacja ASP.NET jest pakowana jako obraz Docker i wdrażana w Kubernetes jako Deployment. Każda nowa wersja jest wdrażana ściągając obraz z Docker Hub.
- **Konfiguracja**: Sekrety, takie jak klucze API oraz hasła do bazy danych są przechowywane jako Secrets.
- **Monitorowanie**: Aplikacja używa paczkę Serilog, która generuje szczegółowe logi.

### 1.4.3 Next.js (Frontend)

- **Opis**: Next.js jest frameworkiem do budowy aplikacji frontendowych, który wspiera zarówno SSR, jak i interaktywność w przeglądarce. Movies Service używa głównie logikę po stronie przeglądarki.
- **Wdrażanie**: Aplikacja Next.js jest budowana jako obraz Docker i wdrażana w Kubernetes jako Deployment. Kubernetes ściąga obraz z Docker Hub.
- **Konfiguracja**: Konfiguracja aplikacji jest zarządzana za pomocą Secrets. Sekrety zawierają jedynie publiczne hasła do brokera MQTT.

## 1.5 Zewnętrzne serwisy

### 1.5.1 Cloudinary (Zarządzanie zdjęciami)

- **Opis**: Cloudinary jest zewnętrznym serwisem do zarządzania, optymalizacji i dostarczania zasobów multimedialnych, takich jak obrazy i filmy. Movies Service używa głównie zarządanie o optymalizacje obrazów.
- **Integracja**: API umożliwia dodawanie zdjęć, na przykład plakaty filmów, i zapisuje te zdjęcia na serwerze Cloudinary. Zapisuje równierz URL do tych zapisanych zdjęć w bazie danych, co potem umożliwia UI na wczytanie tych zdjęć.
- **Konfiguracja**: Dane konfiguracyjne, takie jak API keys i sekrety, są przechowywane w Kubernetes Secrets.
- **Zarządzanie**: Cloudinary Dashboard umożliwia za zarządanie, oraz monitorowanie zasobów.

### 1.5.2 HiveMQ (Brokering MQTT)

- **Opis**: HiveMQ jest zewnętrzną platformą brokeringową MQTT.
- **Integracja**: Frontend przysłuchuje się wielu tematom jakie backend może wysłać, na przykład notyfikacjom. Może równiesz wysyłać wiadomości, aby korzystać z interaktywnego, publicznego czatu. Backend przysłuchuje się tym wiadomościom, aby potwierdzić tożsamość nadawcy.
- **Konfiguracja**: Hasła i hosty do 3 różnych klientów są przechowywane w Secrets. 2 z tych klientów są używane przez frontend, gdzie wszystkie zmienne środowiskowo są widoczne, więc klienci mają ograniczone prawa, na przykład subscribe only. Jest to bezpieczne, ponieważ wszystkie wiadomości wysyłane przez backend nie są przeznaczone dla konkretnych użytkowników.
- **Zarządzanie**: Monitorowanie odbywa się na stronie internetowej HiveMQ.

## 1.6 Konfiguracja i zarządzanie

ConfigMaps głównie konfigurują zmienne środowiskowe dla każdego z deploymentu. W przypaku bazy danych, ConfigMap przechowuje zmienne środowiskowe i wolumeny. Secrets przechowują poufne informacje takie jak klucze API oraz hasła uwierzytelniające. Do zarządzania używane są komendy z CLI kubectl, takie jak:

- **kubectl delete komponent nazwa**: Usuwa komponent.
- **kubectl apply komponent nazwa**: Tworzy komponent.

## 1.7 Zarządzanie błędami

Deployment backend zawiera liveness probe, który zarządza błędami. Backend zawiera endpoint /healthz, który zwraca informacje o tym, czy aplikacja działa poprawnie. Liveness proba wysyła zapytanie co 10 sekund, i jeżeli aplikacja nie jest zdrowa, to pod jest resetowany.

StatefulState bazy danych również zawiera liveness probe. Tak samo jak backend, jeżeli probe wykryje, że aplikacja nie jest zdrowa, pod jest resetowany.

## 1.8 Skalowalność

Skalowanie backendu oraz frontendu jest zapewnione przez HPA, czyli Horizontal Pod Autoscaler. Każdy deployment ma skonfigurowane minimalne oraz maksymalne przydzielone zasoby, a kiedy obciążenie staje się za wysokie, HPA automatycznie tworzy kolejne pody. Kiedy obiążenie jest niższe, HPA, automatycznie usuwa pody. HPA jest skonfigurowany w taki sposób, że stara się, żeby każdy pod używał 50 swojego CPU. Kiedy nie jest to możliwe, HPA tworzy kolejne pody. HPA jest również skonfigurowany w taki sposób, aby liczba replik wynosiła pomiędzy 1 a 20. Backend i frontend używają dwa różne HPA, jednak są one identycznie skonfigurowane. Skalowaność zapewniają również ingressy. Ponieważ użyty jest nginx controller, ingressy są atoamtycznie skalowalne w zależnośći od obciążenia

## 1.9 Wymagania dotyczące zasobów

- **Neo4J**: Zakładając w miarę wysokie obciążenie, jedna replika baza danych powinna wykorzystywać nie więcej niż 2 rdzenie CPU, oraz 7 GB RAM.
- **API**: Ponieważ Kubernetes dynamicznie skaluje backend, pojedyncze repliki wykorzystują stosunkowo niewiele zasobów, pomiędzy 0.1-0.2 CPU oraz 256-512 Mi RAM. Czas odpowiedzi API wynosi przeważnie poniżej 0.1 sekund, prawie zawsze poniżej 0.2 sekund. Wyjątkiem są zapytania, które wymagaja użycia Cloudinary. Czas odpowiedzi w takich przypadkach znacząco się wydłuża.
- **frontend**: Podobnie jak API, frontend będzie skalowany horyzontalnie, jednak pojawiają się problemy techniczne, jeżeli zasoby są za małe. Dlatego więc, każdy pod używa 0.5-1 rdzeniów CPU i 2-3 GB pamięci.

## 1.10 Architektura sieciowa

Każdy serwis, czyli backend-db, backend i frontend, jest typu ClusterIP, co oznacza, że aplikacje mogą komunikować się ze sobą tylko we wnątrz klastra. Ponieważ API oraz frontend muszą być dostępne na zewnątrz klastra, użyty jest ingress, który zapewnia dostęp do tych serwisów. API komunikuje się z bazą danych za pomocą protokoły bolt. Użytownik ma dostęp do API oraz UI dzięki protokołowi HTTP. UI również wysyła zapytania do API protokołem HTTP, jednak używa również MQTT, między innymi do interaktywnego, publicznego czatu, oraz SSE, aby wyświetlać reklamy. NGINX Ingress Controller zarządza ruchem HTTP. Narzędzie te obsługuje przekierowywania, aby UI oraz API były dostępne na zewnątrz klastra. Zarządza również przepustowością, co zapewnia skalowalność.