




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

# E-commerce Unicorn Apparel w Kubernetes

**Alicja Chmura**

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

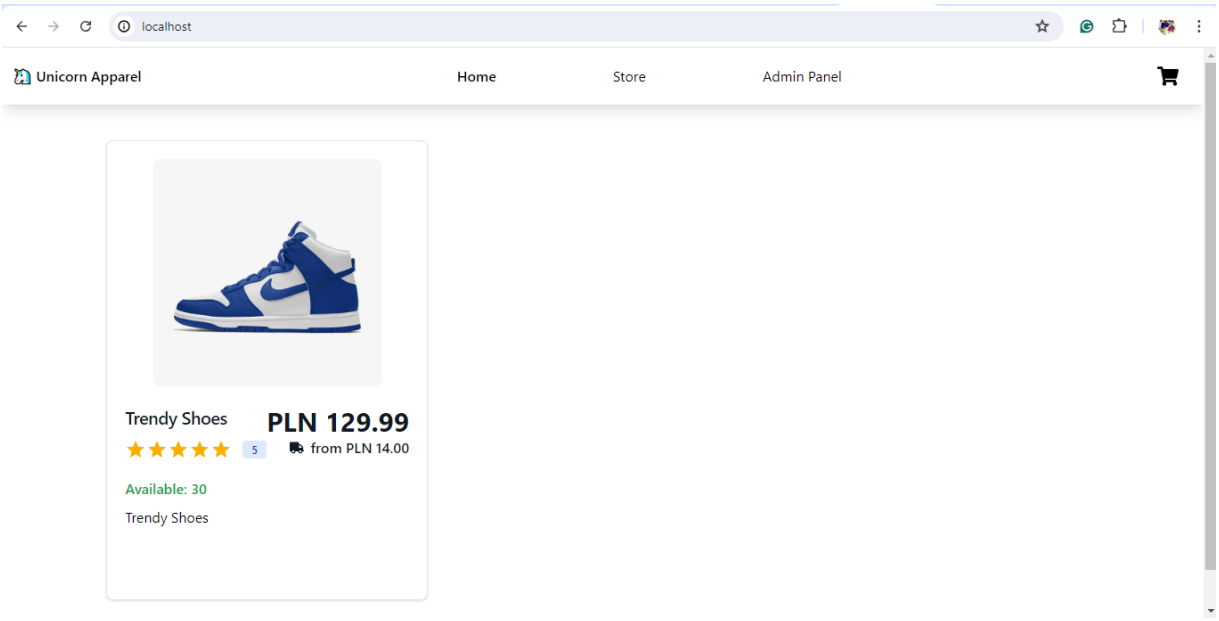
**Gdańsk**

**26 czerwca 2024**

# Spis treści

# 1 Opis projektu

Projekt powstał ze względu na duże zainteresowanie platformą Unicorn Apparel. Celem tego projektu jest zapewnienie solidnej i skalowalnej infrastruktury dla aplikacji e-commerce przy użyciu Kubernetes. Architektura składa się z aplikacji klienckiej napisanej w React Vite, API serwera w ExpressJS oraz bazy danych grafowej Neo4j.




Rysunek 1: Aplikacja uruchomiona w klastrze Kubernetes

## 1.1 Opis architektury - 8 pkt

Aplikacja jest budowana na klastrze Kubernetes, wykorzystując orkiestrację kontenerów do zapewnienia wysokiej dostępności, skalowalności i łatwości zarządzania. Główne komponenty to:

1. Serwis Kliencki: Aplikacja frontendowa dostarczana na porcie 3000.
2. API Serwera: Backendowy serwis komunikujący się z bazą danych Neo4j i obsługujący logikę biznesową.
3. Baza Danych Neo4j: Baza grafowa używana do przechowywania i zapytywania danych.

Kubernetes zarządza tymi komponentami za pomocą Deploymentów (klient i serwer), StatefulSeta (baza Neo4j), Serwisów, ConfigMaps, Secrets, PersistentVolume i PersistentVolumeClaima oraz Ingressa, zapewniając płynną komunikację i skalowalność. Aplikacja dostepna z zewnatrz przez Ingress Nginx wysyła żądania do API backendu uruchomionego w klastrze, a aplikacja backendowa łączy sie z bazą danych i pobiera lub modyfikuje jej dane. Dodatkowo wykorzystane zostały moduły metryki oraz skalowania HorizontalPodAutoscaler (więcej podów zostaje deployowanych w odpowiedzi na zwiększony traffic) umożliwiajace sprawniejsze zarzadzanie klastrem.

## 1.2 Opis infrastruktury - 6 pkt

Aplikacja korzysta z zbudowanych wcześniej i udostępnionych na Docker Hub obrazów. Infrastruktura obejmuje klaster Kubernetes, który może działać na dowolnym dostawcy chmury, takim jak AWS, GCP lub Azure. Klaster zawiera:

- Sieć: Serwisy są wystawiane za pomocą Deploymentów i Ingressa, umożliwiając zarówno dostęp wewnętrzny, jak i zewnętrzny.
- Magazynowanie: Trwałe przechowywanie zarządzane jest za pomocą PersistentVolume i PersistentVolumeClaima, zapewniając niezawodność i dostępność danych.
- Zarządzanie zasobami: Limity i żądania zasobów są ustawiane w celu optymalizacji wykorzystania CPU i pamięci.

## 1.3 Opis komponentów aplikacji - 8 pkt

- Deployment metryki badajacy wykorzystanie zasobów (uruchomiony przed innymi komponentami).
- Klient: Zbudowana aplikacja frontendowa React Vite działająca na porcie 3000. Wdrażana jako Deployment z 3 replikami. Można się dostać do niej z zewnątrz dzięki Ingressowi Nginx.
- Serwer: Serwis backendowy ExpressJS działający na porcie 4000, który łączy się z bazą danych Neo4j. Dane potrzebne do połączenia się z bazą są przechowywane w Secret. Serwer wdrażany jest jako Deployment ze zmienną liczbą replik (od 3 do 6) dobieraną przez HorizontalPodScaler w zależności od trafficu. Loadbalancing jest zapewniony przez Ingress Nginx.
- Neo4j: Baza danych grafowa działająca na portach 7474 (HTTP) i 7687 (protokół Bolt). Jest wdrożona jako StatefulSet, co zapewnia trwałość i stabilność danych. Korzysta także z PersistentVolume i PersistentVolumeClaim, umożliwiających trwałe przechowywanie zebranych danych w lokalnej pamieci masowej.

Każdy komponent jest wdrażany jako osobny pod zarządzany przez Kubernetes, a Serwisy zapewniają stabilne identyfikatory sieciowe.

## 1.4 Konfiguracja i zarządzanie - 4 pkt

Konfiguracja aplikacji jest zarządzana za pomocą ConfigMaps i zmiennych środowiskowych przechowywanych w Secrets. Baza danych Neo4j jest inicjowana za pomocą skryptu przechowywanego w ConfigMap. Kubernetes zarządza wdrażaniem, skalowaniem i aktualizacją aplikacji, minimalizując przestoje. Aplikacje w klastrze sa konfigurowane poprzez pliki yaml. Korzystajac z kubectl w terminalu można usuwać i dodawać pody, zmieniać ich zasoby lub ustawienia oraz sprawdzać ich stan. Aktualizowanie aplikacji odbywa sie poprzez wprowadzenie odpowiednich zmian w pliku .yaml i jego ponowne zastosowanie (kubectl apply -f). Do cofniecia zmian służy komenda kubectl rollout undo.

## 1.5 Zarządzanie błędami - 2 pkt

Zarządzanie błędami obejmuje monitorowanie logów aplikacji, ustawianie livenessProbe i readinessProbe dla podów klienta i serwera. Zużycie zasobów poszczególnych kontenerów jest monitorowane przez Metrics Server, który z kolei przekazuje te dane do HorizontalAutoScaler. Automatyczne zdolności samonaprawcze Kubernetes automatycznie restartują nieudane kontenery.

## 1.6 Skalowalność - 4 pkt

Skalowalność jest zapewniona dzięki użyciu Metrics Server w połączeniu z HorizontalAutoScaler. Kontener Metrics zbiera informacje o wykorzystaniu procesora i pamięci RAM bezpośrednio z kubeletów, co jest kluczowym elementem w procesie skalowania. Ten kontener współpracuje z HorizontalPodAutoscalerem, który umożliwia efektywne zarządzanie zasobami i redukcję ryzyka wystąpienia problemów dzięki regularnemu monitorowaniu zużycia mocy obliczeniowej procesora. Gdy wykryje on zbyt wysokie obciążenie, automatycznie tworzy kolejne repliki aplikacji zgodnie z określonymi w konfiguracji zasadami (od 3 do 6).

## 1.7 Wymagania dotyczące zasobów - 2 pkt

Odgórnie ustalone w plikach konfiguracyjnych wymagania dotyczące zasobów przydzielanych poszczególnym kontenerom:

| | Klient | Serwer | Baza Danych |
|---|---|---|---|
| CPU | 500m | 1000m | 2000m |
| RAM | 1500Mi | 1024Mi | 2048Mi |

Tabela 1: Limity zasobów

Oczekiwania dotyczące wydajności obejmują niski czas odpowiedzi przy zapytaniach do bazy danych oraz wysoką dostępność. Faktyczne zużycie zasobów podczas ruchu na stronie uzyskane dzięki metrykom:

| | **Klient** | **Serwer** | **Baza Danych** |
|---|---|---|---|
| **Zużycie RAM** | 800Mi | 700Mi | 500Mi |
| **Zużycie CPU** | 5m | 400m | 200m |
| **Miejsce na dysku** | 1.5Gi | 1.5Gi | 1Gi |
| **Czas odpowiedzi** | $< 2$s | $< 400$ms | $< 3$s |

Tabela 2: Zużycie zasobów i czas odpowiedzi

## 1.8 Architektura sieciowa - 4 pkt

Klastr Kubernetes został skonfigurowany z wykorzystaniem protokołu sieciowego CNI (Container Network Interface), który umożliwia każdemu kontenerowi w klastrze komuni-

kację sieciową. W moim klastrze używam CNI pluginu, który zapewnia elastyczne zarządzanie adresami IP oraz konfigurację routingu między kontenerami.

### Wykorzystywane Protokoły:

1. **HTTP**: Protokoły te są używane do komunikacji między klientem a serwerem e-commerce oraz do dostępu do aplikacji poprzez Ingress.
2. **TCP**: Wykorzystywany głównie do komunikacji między kontenerami, na przykład pomiędzy frontendem (client) a backendem (server), oraz do komunikacji z bazą danych.
3. **Bolt**: Protokół Bolt jest wykorzystywany do komunikacji między serwerem aplikacyjnym a bazą danych Neo4j, zapewniając szybki i efektywny dostęp do danych.

### Narzędzia do Zarządzania Siecią:

1. **Kubernetes Networking**: Zarządzanie siecią w klastrze Kubernetes obejmuje konfigurację Ingress, Service oraz Network Policies, które definiują reguły komunikacji między usługami oraz zarządzają dostępem do aplikacji.
2. **NGINX Ingress Controller**: Kontroler Ingress NGINX zarządza ruchem sieciowym do aplikacji w klastrze, umożliwiając elastyczne routowanie ruchu HTTP do różnych usług w klastrze (klienta i serwera).

## 1.9 Podsumowanie

Dzięki powyższej strukturze i wykorzystaniu potężnych możliwości orchestracji Kubernetes, aplikacja zapewnia wysoką dostępność, skalowalność oraz efektywne zarządzanie zasobami. Każdy komponent jest konteneryzowany i zarządzany niezależnie, co zapewnia elastyczność i odporność całego systemu.

# Literatura

[1] Kubernetes Authors, *Kubernetes documentation*, Kubernetes.io (2023).

[2] A. Burns, B. Grant, and D. Oppenheimer, *Kubernetes: Up and running*, O'Reilly Media, Sebastopol, CA, 2020.

[3] V. Kozlov and P. Morrice, *Kubernetes: A container scheduler for cloud computing*, Proceedings of the 6th International Conference on Cloud Computing, ACM, 2022, pp. 112–120.

[4] R. Sharma and S. Kumar, *Security challenges and considerations in kubernetes*, Tech. Report TC-2021-123, TechCorp, 2021.

[5] D. Smith, *Understanding kubernetes: A beginner's guide*, 2019.