



Uniwersytet Gdański
Wydział Matematyki, Fizyki i Informatyki
Instytut Informatyki

Facebook Clone

Łukasz Nowosielski

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

Gdańsk
8 czerwca 2024

Spis treści

1	Opis projektu	2
1.1	Opis architektury - 8 pkt	2
1.2	Opis infrastruktury - 6 pkt	2
1.3	Opis komponentów aplikacji - 8 pkt	3
1.4	Konfiguracja i zarządzanie - 4 pkt	3
1.5	Zarządzanie błędami - 2 pkt	4
1.6	Skalowalność - 4 pkt	4
1.7	Wymagania dotyczące zasobów - 2 pkt	4
1.8	Architektura sieciowa - 4 pkt	4

1 Opis projektu

Projekt "facebook-clone" powstał z myślą o dynamicznie rozwijającej się firmie **ConnectSphere**, która specjalizuje się w dostarczaniu innowacyjnych rozwiązań technologicznych. **ConnectSphere** zauważyła, że wiele małych i średnich przedsiębiorstw potrzebuje prostych, ale skutecznych narzędzi do budowania społeczności i komunikacji z klientami. Jednak popularne platformy społecznościowe, takie jak Facebook, często są zbyt obciążone nadmiarem funkcji, co prowadzi do wolnej i nieefektywnej komunikacji.

ConnectSphere potrzebowała prostego, ale skutecznego narzędzia do budowania społeczności wokół swojej misji. Chcieli stworzyć miejsce, gdzie firmy mogłyby łatwo komunikować się ze swoimi klientami, dzielić się aktualnościami i informacjami o produktach, a także otrzymywać bezpośrednie opinie.

Dlatego powstał pomysł na stworzenie "facebook-clone platformy, która oferuje podstawowe funkcje społecznościowe, takie jak tworzenie profilu, udostępnianie postów, komentowanie i reagowanie na posty innych użytkowników. Projekt ma na celu dostarczenie tych funkcji w prosty i intuicyjny sposób, umożliwiając **ConnectSphere** skuteczną komunikację i budowanie społeczności wokół ich produktów i usług.

1.1 Opis architektury - 8 pkt

"facebook-clone" jest zbudowane z interfejsu użytkownika (frontend), serwera aplikacji (backend), bazy danych oraz serwisu uwierzytelniania Keycloak.

Aplikacja jest hostowana na klastrze Kubernetes, który zapewnia skalowalność, niezawodność i wydajność. Interfejs użytkownika jest dostępny dla użytkowników za pośrednictwem przeglądarki internetowej, podczas gdy serwer aplikacji komunikuje się z bazą danych MongoDB i obsługuje żądania użytkowników. Elementy te są zabezpieczone dzięki integracji z Keycloak.

Aplikacja korzysta z Ingress w Kubernetes, który zarządza dostępem do usług w klastrze, zapewniając wydajne i niezawodne przekierowanie ruchu. Dzięki Ingress, aplikacja jest dostępna pod adresem <https://facebook-clone.com> (frontend), <https://api.facebook-clone.com> (backend) i <https://keycloak.facebook-clone.com> (keycloak). Ingress umożliwia także łatwe zarządzanie ruchem sieciowym, co jest kluczowe dla utrzymania wydajności i niezawodności aplikacji.

1.2 Opis infrastruktury - 6 pkt

Aplikacja "facebook-clone" jest zaprojektowana do działania w środowisku Kubernetes, wykorzystując Minikube. Minikube to narzędzie, które umożliwia uruchomienie jednego węzła klastra Kubernetes na lokalnym komputerze. Jest to idealne rozwiązanie dla deweloperów, którzy chcą testować swoje aplikacje w środowisku Kubernetes bez konieczności tworzenia pełnego klastra. Specyfikacja klastra Minikube to 4GB pamięci RAM, 2 vCPU i 10GB miejsca na dysku.

1.3 Opis komponentów aplikacji - 8 pkt

Aplikacja składa się z:

- **Frontend:** Interfejs użytkownika został zbudowany za pomocą biblioteki React. Do serwowania statycznych plików używamy serwera Nginx. Kod źródłowy frontendu jest zawarty w kontenerze Docker, który jest konfigurowany i uruchamiany za pomocą Dockerfile.
- **Backend:** Backend został zbudowany za pomocą frameworka Express.js. Backend komunikuje się z bazą danych MongoDB, która jest hostowana na platformie MongoDB Atlas. Podobnie jak frontend, backend jest również zawarty w kontenerze Docker, który jest konfigurowany i uruchamiany za pomocą Dockerfile.
- **Baza danych:** Aplikacja korzysta z MongoDB jako bazy danych. Jest ona hostowana w chmurze, co zapewnia dodatkową niezawodność i elastyczność. Baza danych jest skonfigurowana do komunikacji z serwerem aplikacji za pośrednictwem bezpiecznego połączenia.
- **Keycloak:** Do zarządzania uwierzytelnianiem i autoryzacją w aplikacji używamy Keycloak. Keycloak jest skonfigurowany do pracy w trybie "client credentials", gdzie autoryzacja jest oparta na identyfikatorze i tajnym kluczu klienta, a nie na danych użytkownika. Tokeny uwierzytelniające są generowane przez Keycloak i przekazywane do backendu, gdzie są używane do zabezpieczania naszego API.
- **PostgreSQL:** Aplikacja korzysta z bazy danych PostgreSQL, która jest zarządzana przez Keycloak. Baza danych jest przechowywana na stałym woluminie (PVC), co pozwala na zachowanie danych oraz ustawień dla Keycloak nawet po zatrzymaniu i ponownym uruchomieniu kontenera.

1.4 Konfiguracja i zarządzanie - 4 pkt

Konfiguracja i zarządzanie aplikacją na poziomie klastra Kubernetes odbywa się na kilka sposobów:

- **Deployments:** Każdy komponent aplikacji (frontend, backend, Keycloak, baza danych PostgreSQL) jest uruchamiany jako osobny Deployment w Kubernetes. Deployments pozwalają na łatwe skalowanie i aktualizacje komponentów aplikacji.
- **Serwisy:** Komunikacja między komponentami aplikacji jest zarządzana za pomocą Serwisów Kubernetes, które zapewniają odkrywanie i balansowanie obciążenia.
- **ConfigMaps i Secrets:** Konfiguracja aplikacji jest przechowywana w ConfigMaps (Ingress, Keycloak, PostgreSQL) i Secrets (Backend, PostgreSQL). Pozwala to na centralne zarządzanie konfiguracją i bezpieczne przechowywanie poufnych danych.
- **Persistent Volume Claims:** Dane bazy danych PostgreSQL są przechowywane na Persistent Volume za pomocą Persistent Volume Claim, co zapewnia trwałość danych.
- **Ingress:** Dostęp do aplikacji z zewnątrz klastra jest zarządzany za pomocą Ingress, który kieruje ruch sieciowy do odpowiednich Serwisów.

1.5 Zarządzanie błędami - 2 pkt

- **Obsługa błędów na poziomie aplikacji:** W backendzie i frontendzie korzystam z bloków `try catch` do przechwytywania i obsługi błędów. Dzięki temu, nawet w przypadku nieoczekiwanych błędów, aplikacja nie przestaje działać.
- **Zarządzanie błędami na poziomie infrastruktury:** W konfiguracji Deployment dla backendu, korzystam z mechanizmu replikacji Kubernetes. Dzięki temu, jeśli jeden z podów aplikacji napotka błąd i przestanie działać, Kubernetes automatycznie przekieruje ruch do drugiego poda.

1.6 Skalowalność - 4 pkt

Skalowalność jest kluczowa w architekturze aplikacji opartej na Kubernetes.

- **Skalowanie horyzontalne:** Aplikacja jest skalowana horyzontalnie poprzez zwiększanie liczby replik podów na backendzie. Obecnie liczba replik na backendzie została zwiększona do 2, co pozwala na lepsze rozłożenie obciążenia i zwiększa przepustowość. Kubernetes automatycznie rozprowadza ruch między replikami, zapewniając równomierne obciążenie.

1.7 Wymagania dotyczące zasobów - 2 pkt

- **Frontend:** 300MB pamięci RAM, 0.2 rdzenia CPU
- **Backend:** 600MB pamięci RAM, 0.2 rdzenia CPU (na replike)
- **PostgreSQL:** 256MB pamięci RAM, 0.15 rdzenia CPU
- **Keycloak:** 2000MB pamięci RAM, 1 rdzeń CPU

1.8 Architektura sieciowa - 4 pkt

Architektura sieciowa aplikacji opiera się na modelu sieciowym Kubernetes, który umożliwia komunikację między podami oraz z zewnętrznymi usługami sieciowymi.

- **Ingress:** Wszystkie usługi (frontend, backend, Keycloak) są dostępne poprzez Ingress, który zarządza dostępem do usług w klastrze z zewnątrz. Ingress umożliwia routowanie ruchu sieciowego do odpowiednich usług na podstawie adresu URL i metody HTTP.
- **Protokoły:** Aplikacja korzysta z protokołu HTTP do komunikacji między frontendem a backendem. Do zarządzania stanem sesji i autoryzacji używany jest protokół OAuth2 z Keycloak.