Uniwersytet Gdański
Wydział Matematyki, Fizyki i Informatyki
Instytut Informatyki

Aplikacja do gry w Gomoku

Dawid Kalinowski

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

Gdańsk

27 czerwca 2024

Spis treści

1 Opis projektu

Firma zajmujaca sie grami planszowymi, Board Games Interactive, zleciła stworzenie aplikacji webowej, która bedzie docelowo obsługiwała rozgrywki w przeróżnych grach planszowych, miedzy graczami z całego świata. Na sam poczatek przedsiebiorstwo postanowiło zapewnić graczom możliwość gry w gre "Gomoku". Serwis ma działać w przegladarce, zarówno na komputerach, jak i na urzadzeniach mobilnych. System musi zapewnić płynna rozgrywke dla tysiecy graczy jednocześnie.

1.1 Opis architektury - 6 pkt

Architektura aplikacji została zaimplementowana w środowisku Kubernetes, co w przyszłości umożliwi łatwiejsze skalowanie aplikacji oraz wdrażanie nowych zmian. Składa sie z trzech oddzielnych komponentów: Backendu w Node.js, Frontendu w React.js oraz bazy danych MongoDB. Każdy z nich jest osobnym kontenerem w klastrze Kubernetes.

1.2 Opis infrastruktury - 6 pkt

Aplikacja korzysta z platformy Kubernetes, oraz przechowuje obrazy Dockerowe na platformie DockerHub. Frontend i backend używaja LoadBalancer do komunikowania sie z innymi klastrami w Kubernetesie.
Backend używa Node.js, biblioteki Express, komunikuje sie z baza danych za pomoca Mongoose. Używa również WebSocketów (biblioteka ws) oraz JWT.
Frontend używa frameworka React.js, a zarzadzanie sesja użytkownika odbywa sie za pomoca js-cookie.
Baza danych używa ClusterIP do komunikowania sie z innymi klastrami w Kubernetesie.

1.3 Opis komponentów aplikacji - 6 pkt

Aplikacja składa sie z trzech głównych komponentów:

- **Backend (Node.js):** Backend aplikacji został wyprodukowany w Node.js przy użyciu frameworka Express.js. Odpowiada za obsługe użytkowników oraz gier, miedzy innymi poprzez pobieranie oraz wysyłanie informacji z / do bazy danych w MongoDB. Backend udostepnia RESTful API, które obsługuje operacje takie jak rejestracja, logowanie użytkowników, tworzenie, dołaczanie do gier, w przypadku Gomoku również stawianie pionków na planszy. Panel administratora udostepnia również takie funkcjonalności, jak edycja poszczególnych instancji gier, manipulowanie uprawnieniami oraz danymi zwykłych użytkowników. Do zarzadzania stanem sesji użytkowników zastosowano tokeny JWT, a logowanie użytkowników jest zabezpieczone funkcja hahsujaca, a dokładniej bcrypt. Aplikacja umożliwia również czatowanie z innymi użytkownikami na wspólnym forum, za pomoca protokołu MQTT.
- **Frontend (React.js):** Frontend aplikacji został zaimplementowany w bibliotece React.js. Użytkownik może, korzystajac z intuicyjnych komponentów, takich jak pola tekstowe, przyciski, poruszać sie po aplikacji, w szczególności dołaczać do gier oraz grać. Aplikacja posiada przejrzysta plansze do gry w Gomoku, która po zmianie

stanu planszy na bieżaco sie odświeża. Frontend komunikuje sie z Backendem za pomoca zapytań HTTP, przesyłajac i odbierajac dane w formacie JSON. Aplikacja jest również przystosowana pod urzadzenia mobilne, uzależniajac rozmiar poszczególnych komponentów od maksymalnej rozdzielcości okna.

- **Baza danych MongoDB:** Do przechowywania informacji o wszystkich użytkownikach, oraz toczonych grach w aplikacji, wykorzystano MongoDB, baze danych NoSQL. Baza przechowuje dane użytkowników, informacje zarówno o wszystkich aktualnie toczonych meczach, jak i historycznych gier.

Każdy komponent jest skonfigurowany w osobnym pliku YAML dla Kubernetes i jest wdrażany jako oddzielny pod.

1.4 Konfiguracja i zarzadzanie - 4 pkt

Konfiguracja aplikacji odbywa sie poprzez pliki YAML dla Kubernetes, które definiuja deploymenty i serwisy. Deployment zarówno backendu, jak i frontendu, definiuje obraz Docker, m.in. liczbe replik - w tym przypadku 3 - oraz zmienne środowiskowe, w tym URI do bazy danych MongoDB. Usługa zapewnia zewnetrzny dostep do backendu i frontendu poprzez LoadBalancer. Baza danych również jest w 3 replikach, a komunikacja odbywa sie za pomoca ClusterIP.

1.5 Zarzadzanie błedami - 2 pkt

Aplikacja obsługuje błedy róznych zapytań, i wszelkie nieudane / nieprawidłowe działania wypisuje w konsoli użytkownika.

1.6 Skalowalność - 4 pkt

Za skalowalność aplikacji jest odpowiedzialny Horizontal Pod Autoscaler (HPA). Jest skonfigurowany do monitorowania backendu, i zmienia liczbe jego replik w zakresie liczb 3-10, w zależności od obciażenia procesora. Aktualnie jego średnie zużycie jest ustawione na poziomie 60

1.7 Wymagania dotyczace zasobów - 2 pkt

Wymagania dotyczace zasobów sa nastepujace:

- **Backend:** Minimalne: 0.5 CPU, 512MB RAM, Maksymalne: 1 CPU, 1 GB RAM
- **Frontend:** Minimalne: 0.25 CPU, 256MB RAM, Maksymalne: 0.5 CPU, 512MB RAM
- **Baza danych:** Minimalne: 0.5 CPU, 1GB Ram, Maksymalne: 1CPU, 2GB Ram

Wszystkie powinny wykonywać sie w akceptowalnym dla użytkownika czasie, poniżej 100 ms.

1.8 Architektura sieciowa - 4 pkt

Wszystkie serwisy w klastrze komunikuja sie ze soba w sieci lokalnej localhost. Serwisy sa skonfigurowane na działanie w nastepujacych portach:
- Backend nasłuchuje na porcie 3003
- Frontnend na porcie 3000
- Baza danych na porcie 27017

Literatura

[1] *Dokumentacja express js https://expressjs.com/.*

[2] *Dokumentacja js-cookie https://www.npmjs.com/package/js-cookie.*

[3] *Dokumentacja kubernetes https://kubernetes.io/docs/home/.*

[4] *Dokumentacja mongodb https://www.mongodb.com/docs/.*

[5] *Dokumentacja mongoose https://mongoosejs.com/docs/.*

[6] *Dokumentacja react https://legacy.reactjs.org/docs/getting-started.html.*