Uniwersytet Gdański
Wydział Matematyki, Fizyki i Informatyki
Instytut Informatyki

Aplikacja do składania zamówień - kiosk McDonald's

Kacper Hołowaty

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

Gdańsk

20 czerwca 2024

Spis treści

1 Opis projektu

Pewna mała restauracja, sprzedająca fast foody, zauważyła, że w związku z tym, że konkurencja wprowadziła do swoich restauracji interaktywne ekrany, umożliwiające klientom składanie zamówień bez potrzeby podchodzenia do kas i rozmawiania z kimkolwiek z pracowników, to spadły im wyniki sprzedażowe. Nie chcieli więc pozostać w tyle, bo pomimo niekiedy lepszego jedzenia, często klienci omijali ich restaurację z uwagi właśnie na brak możliwości zamówienia jedzenia w aplikacji.

Dlatego też powstała aplikacja do składania zamówień - kiosk McDonald's. Klienci mogą, po kliknięciu w ekran wybrać jaki produkt chcą zamówić, wybrać ilość i jakie dodatkowe składniki mają być dodane do ich zamówienia. Mogą oni następnie przejść do koszyka i edytować tam swoje zamówienie oraz ostatecznie zapłacić, po czym otrzymają oni paragon i numer swojego zamówienia.

Pracownicy restauracji, po uprzednim zalogowaniu się w panelu administratora, mogą przejrzeć statystyki sprzedaży z podziałem na dzień i miesiąc. Natomiast administrator może dodatkowo edytować produkty dostępne w menu.

1.1 Opis architektury - 8 pkt

Aplikacja posiada cztery podstawowe serwisy: interfejs użytkownika (frontend), serwer aplikacji (backend), bazę danych oraz serwis Keycloak, zarządzający dostępem w aplikacji.

Aplikacja jest hostowana na klastrze Kubernetes, co zapewnia jej niezawodność. Interfejs użytkownika jest dostępny pod adresem **http://localhost:32000** w przeglądarce internetowej. Serwer aplikacji komunikuje się z bazą danych MongoDB i obsługuje rozmaite żądania CRUD od użytkowników. Keycloak natomiast, za zadanie ma chronić nieupoważnionych użytkowników przed dostępem do panelu administracyjnego aplikacji.

1.2 Opis infrastruktury - 6 pkt

Używam klastra Kubernetes w wersji v1.29.2 uruchomionego w Docker Desktop.

1.3 Opis komponentów aplikacji - 8 pkt

Aplikacja składa się z następujących komponentów:

- **Frontend:** Interfejs użytkownika został stworzony przy użyciu frameworka React.js. Korzysta z backendu, wysyłając różne żądania.
- **Backend:** Backend został zbudowany za pomocą Express.js. Komunikuje się z bazą danych MongoDB, która jest hostowana lokalnie.
- **Baza danych:** Wykorzystywana jest baza MongoDB. Przechowywane są w niej informacje o produktach dostępnych w McDonald's, ale także historia wszystkich zamówień, co umożliwia stworzenie statystyk. Baza przechowywana jest na stałym woluminie Persistent Volume, co zapewnia trwałość danych.

- **Keycloak:** Keycloak został uruchomiony przy pomocy klastra Kubernetes, a jego konfiguracja zapisana jest na stałym woluminie. W mojej aplikacji posłużył do zabezpieczenia frontendu, gdzie za zadanie miał chronić panel administracyjny przed dostępem dla niektórych użytkowników.

1.4 Konfiguracja i zarządzanie - 4 pkt

Konfiguracja i zarządzanie odbywa się za pomocą:

- **Deploymenty:** Każdy komponent aplikacji (frontend, backend, Keycloak, baza danych MongoDB) jest uruchamiany jako osobny Deployment w Kubernetes. Deploymenty pozwalają na łatwe skalowanie i aktualizację komponentów aplikacji.
- **Serwisy:** Komunikacja między komponentami aplikacji jest zarządzana za pomocą serwisów Kubernetes, które zapewniają odkrywanie i balansowanie obciążenia. W utworzonym przeze mnie klastrze posiadam łącznie uruchomione cztery serwisy (frontend, backend, MongoDB oraz Keycloak).
- **Persistent Volume Claims:** Dane z bazy MongoDB oraz konfiguracja Keycloak'a są przechowywane na Persistent Volume za pomocą Persistent Volume Claim. Zapewniona jest w ten sposób trwałość danych.

1.5 Zarządzanie błędami - 2 pkt

Na poziomie aplikacji

Na frontendzie i backendzie korzystam z konstrukcji: try catch oraz promisów do przechwytywania i obsługi błędów. W przypadku nieoczekiwanych błędów, aplikacja wciąż ma prawo działać.

Na poziomie infrastruktury

W konfiguracji Deployment dla backendu, korzystam z mechanizmu replikacji Kubernetes. Dzięki temu, jeśli któryś z podów aplikacji wyrzuci błąd i przestanie działać, to Kubernetes automatycznie stworzy nowego poda i przekieruje do niego ruch.

1.6 Skalowalność - 4 pkt

Skalowalność jest kluczowa w architekturze aplikacji opartej na Kubernetes. Aplikacja może być skalowana zarówno horyzontalnie jak i wertykalnie.

Skalowanie horyzontalne

Polega ono na zmianie liczby replik (instancji) podów na podstawie obciążenia CPU lub innych zdefiniowanych metryk. Konfiguracja jest oparta na metrykach, takich jak średnie zużycie CPU lub niestandardowe metryki dostarczone przez aplikację.

Skalowanie wertykalne

Polega ono na dostosowaniu rozmiaru zasobów (CPU, pamięć) przypisanych do pojedynczych podów na podstawie rzeczywistego zużycia tych zasobów. Pomaga zminimalizować marnowanie zasobów poprzez dokładne dopasowanie zasobów do potrzeb aplikacji.

1.7 Wymagania dotyczące zasobów - 2 pkt

- **Backend:** 500MB pamięci RAM, 0.2 rdzenia CPU
- **MongoDB:** Persistent Volume Claim dla bazy danych żądą minimum 5GB przestrzeni dyskowej.
- **Keycloak:** Persistent Volume Claim dla Keycloaka żądą minimum 1GB przestrzeni dyskowej.

1.8 Architektura sieciowa - 4 pkt

Po uruchomieniu klastra Kubernetes, architektura sieciowa wygląda następująco: frontend działa w przeglądarce pod adresem **http://localhost:32000**, backend, który bezpośrednio komunikuje się z bazą MongoDB, no i jest swego rodzaju API dla frontendu, działa pod adresem **http://localhost:32001**. Keycloak dostępny jest natomiast pod adresem **http://localhost:32002**. Do komunikacji frontendu z backendem użyty został protokół HTTP. Do autoryzacji używany jest OAuth2 z Keycloakiem.