Uniwersytet Gdański
Wydział Matematyki, Fizyki i Informatyki
Instytut Informatyki

Panda Ramen Sushi

Cezary Prajwowski

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

Gdańsk

26 czerwca 2024

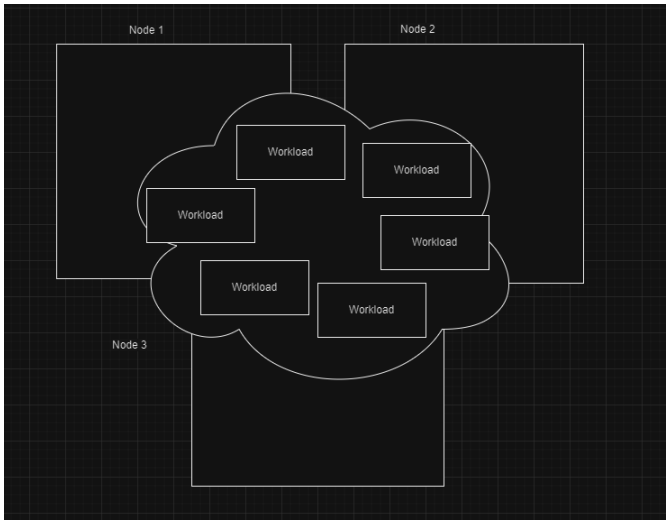
Spis treści

1 Opis projektu

Nowopowstała, rozwijajaca sie restauracja z jedzeniem japońskim potrzebuje strony do składania zamówień, wyświetlania menu restauracji oraz do ogłaszania aktualności. Ma być to miejsce pomocne dla klienta jak i dla właściciela restauracji.

1.1 Opis architektury - 8 pkt

Aplikacja jest wdrożona w klastrze Kubernetes, który odbierajac głównym Node'm zapytanie dotyczace stworzenia workloadu, wysyła je do Workera, który nastepnie za pomoca Dockera uruchamia kontener w sobie. W naszym przypadku mamy 4 takie moduły, SPA (react) dostarczajacy interfejs użytkowinka, API (Express) interakcja miedzy warstwami, DB (Mongo) do zarzadzania baza danych oraz authorization server (Keycloak) do autoryzacji i autentykacji użytkownika. Każda z nich jako Deployment (zarzadzajacy podami) + Service (dajacy dostep do komunikacji do/z poda).



1.2 Opis infrastruktury - 6 pkt

Aplikacja działaja w kontenerach w klastrze Kubernetes. Z narzedzi używamy Kubernetes do orkiestracji kontenerów, który daje środowisko do odpalenia ich w wiekszej ilości i Docker do tworzenia ich. Z 1Gi pamieci masowej korzysta pod bazy danych (mymongo).

1.3 Opis komponentów aplikacji - 8 pkt

- **Frontend**
  - Obraz zawierajacy aplikacje React.
  - Wdrożony jako jedno-replikowy deployment.
  - Service typu LoadBalancer, dostepny na zewnatrz pod `localhost:3000`.
  - Nie zawiera żadnych zmiennych środowiskowych.

- **Backend**
  - Aplikacja Express na obrazie `node:alpine`.
  - Wdrożona jako auto-skalujacy deployment.
  - Service typu NodePort, dostepny dla innych kontenerów pod URL `mybackend:8000`.
  - Zmienne środowiskowe w konfiguracji:
    - `DB_HOST_URL` - nazwa serwisu bazy danych.
    - `KEYCLOAK_HOST_URL` - nazwa serwisu Keycloak.
- **Database (MongoDB)**
  - Baza danych oparta na obrazie `mongo`, z dodanym bazowym `mongoimportem`.
  - Wdrożona jako StatefulSet (2 repliki).
  - Service z nazwa domeny wewnatrz sieci Kubernetes `mymongo-set`.
  - Zmienne środowiskowe:
    - `MONGO_INITDB_ROOT_USERNAME` - implementowane dzieki `mongo-secret`.
    - `MONGO_INITDB_ROOT_PASSWORD` - implementowane dzieki `mongo-secret`.
- **Keycloak**
  - Serwer autoryzujacy i uwierzytelniajacy.
  - Wdrożony jako jedno-replikowy deployment.
  - Przekazane dane w formie secret dotyczace danych wrażliwych:
    - Hasło Keycloak.
    - Hasło PostgreSQL.
  - Przekazane dane z ConfigMap:
    - Porty.
    - Hosty.
    - Nazwy użytkownika.
  - Service typu LoadBalancer:
    - Dostepny lokalnie na `localhost:8080`.
    - Dostepny dla innych kontenerów w wewnetrznej sieci na `keycloak-service:8080`.
- **Postgres**
  - Baza danych PostgreSQL zamiast defaultowej H2 keycloaka, wdrożona jako Deployment i Service.

1.4 Konfiguracja i zarzadzanie - 4 pkt

Klastry wdrażane sa za pomoca plików .yaml, które inicjalizuja Deploymenty, Serwisy, configmapy, secrety oraz metryki. Kubelet zajmuje sie zarzadzaniem i utrzymywaniem poda. Konfiguracja jest wdrażana za pomoca config-mapów oraz secret‘ów. Zarzadzanie aplikacja głównie komendami kubectl obsługiwanymi przez kubeleta.

1.5 Zarzadzanie błedami - 2 pkt

Kubelet zajmuje sie sprawdzaniem czy kontener jest zdrowy. Pody automatycznie sie restartuja w razie wystapienia awarii. Aby dodatkowo to zabezpieczyć do deploymentu backendu dodałem sondy livenessProbe oraz readinessProbe. Pierwsza sprawdza stan aplikacji i jeśli stwierdzi, że aplikacja nie działa poprawnie to ja zrestartuje. Dopóki druga sonda readinessProbe nie potwierdzi, że aplikacja jest gotowa, Kubernetes nie przesyła do niej ruchu, co zapobiega niedziałajacym żadaniom.

1.6 Skalowalność - 4 pkt

Skalowanie jest wdrożone przy użyciu narzedzia HPA (HorizontalPodAutoscaler) na deploymencie backendu, który jest najbardziej narażony na problemy zwiazane z nagłym przypływem żadań. Polega on na badaniu średniego obciażenia CPU aplikacji, gdy przekroczy 50% startuje kolejny pod, aby rozłożyć ruch na kilka replik. Maksymalnie ustawione na 10 replik.

1.7 Wymagania dotyczace zasobów - 2 pkt

Powinna zawierać informacje na temat wymagań dotyczacych zasobów dla każdego komponentu aplikacji, takie jak ilość pamieci RAM, CPU, miejsce na dysku, itp. Należy również opisać, jakie sa oczekiwania dotyczace wydajności i czasu odpowiedzi dla aplikacji.

Dane zużycia RAM oraz CPU w tabeli pochodza z komendy kubectl top pods. Mierzone były podczas normalnego użytkowania przez jednego użytkownika strony internetowej.

Serwis	**Time (seconds)**	**CPU**	**RAM**
Frontend	0.002s	3m	431Mi
Backend	0.008s	3m	35Mi
MongoDB (1x set)	0.001s	5m	217Mi
Keycloak	0.024s	3m	635Mi

Tabela 1: Czasy odpowiedzi (testowane przez curl adres)

Mongo rezerwuje 512Mi RAM oraz 500m i ma taki sam limit ze wzgledu na zasade guaranteed Quality of Service and guaranteed performance.

Komponent	**Rezerwacja**	**Limit**
Backend (CPU / RAM)	100m / 124Mi	1000m / 1024Mi
MongoDB (CPU / RAM)	500m / 512Mi	500m / 512Mi

Tabela 2: Rezerwacja i limity zasobów dla backendu i MongoDB w Kubernetes

Dodatkowo 1Gi miejsca na dysku dla MongoDB.

1.8 Architektura sieciowa - 4 pkt

Mikroserwisy komunikuja sie za pomoca wewnetrznej sieci. Na zewnatrz na adres localhost:3000 za pomoca LoadBalancera jest wystawiona aplikacja React, która czerpie z innych serwisów, wysyłajac zapytania routowane przez sieć wewnetrzna klastra na przypisane im domeny. Wykorzystane protokoły to TCP/IP oraz HTTP.

Literatura

https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale-walkthrough/