Uniwersytet Gdański
Wydział Matematyki, Fizyki i Informatyki
Instytut Informatyki

Konfiguracja klastra obliczeniowego dla aplikacji typu forum

Piotr Stróżyk

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

Gdańsk

26 czerwca 2024

Spis treści

1 Opis projektu **2**

1.1 Opis architektury - 8 pkt 2

1.1.1 Backend 2

1.1.2 Frontend 4

1.1.3 Baza Danych 5

1.2 Opis infrastruktury - 6 pkt 7

1.3 Opis komponentów aplikacji - 8 pkt 9

1.4 Konfiguracja i zarzadzanie - 4 pkt 10

1.5 Zarzadzanie błedami - 2 pkt 11

1.6 Skalowalność - 4 pkt 12

1.7 Wymagania dotyczace zasobów - 2 pkt 13

1.8 Architektura sieciowa - 4 pkt 14

1 Opis projektu

Firma TechForum Inc., dynamicznie rozwijajaca sie w branży technologicznej, zauważyła rosnaca potrzebe stworzenia platformy umożliwiajacej wymiane wiedzy oraz doświadczeń miedzy specjalistami IT. W odpowiedzi na te potrzebe, powstał projekt MyForum – nowoczesna aplikacja forumowa.

ForumConnect to prosta, lecz funkcjonalna aplikacja oparta na technologii vite-react, z backendem zbudowanym na expressie oraz baza danych mongodb. Głównym celem projektu jest stworzenie intuicyjnej i przyjaznej dla użytkownika platformy, na której profesjonaliści z branży IT moga zadawać pytania, udzielać odpowiedzi oraz prowadzić dyskusje na tematy zwiazane z technologia. Dzieki zastosowaniu nowoczesnych technologii, aplikacja jest szybka, responsywna i łatwa w utrzymaniu.

Projekt ma na celu zaspokojenie rosnacego zapotrzebowania na wysokiej jakości fora tematyczne, wspierajace rozwój zawodowy i wymiane wiedzy miedzy specjalistami IT na całym świecie.

1.1 Opis architektury - 8 pkt

Architektura aplikacji ForumConnect opiera sie na Kubernetes, co umożliwia zarzadzanie klastrem kontenerów w sposób efektywny i skalowalny.

Klastry Kubernetes to zestaw wezłów, które uruchamiaja aplikacje konteneryzowane. Konteneryzacja aplikacji pakuje aplikacje z jej zależnościami i niezbednymi usługami. Sa one lżejsze i bardziej elastyczne niż maszyny wirtualne. W ten sposób klastry Kubernetes pozwalaja na łatwiejsze rozwijanie, przenoszenie i zarzadzanie aplikacjami. Konteneryzacja to proces wdrażania i uruchamiania oprogramowania, który łaczy kod aplikacji ze wszystkimi plikami i bibliotekami potrzebnymi do jej uruchomienia na dowolnej infrastrukturze. Kubernetes to otwartoźródłowe oprogramowanie do orkiestracji kontenerów, za pomoca którego można zarzadzać, koordynować i planować kontenery na duża skale. Kubernetes umieszcza kontenery w podach i uruchamia je na wezłach. Klastr Kubernetes ma, co najmniej, wezeł główny uruchamiajacy pod kontenera oraz płaszczyzne sterowania, która zarzadza klastrem. [1]

Klastr Kubernetes dla tej aplikacji wdraża trzy mikroserwisy.

1.1.1 Backend

Deployment backendu uruchamia jedna instancje aplikacji backendowej dla forum, która komunikuje sie z baza danych MongoDB (adres bazy danych jest przekazywany przez zmienna środowiskowa). Deployment monitoruje zdrowie aplikacji za pomoca sond i zarzadza zasobami, które aplikacja może zużywać.

Service backendu umożliwia innym podom w klastrze (oraz potencjalnie zewnetrznym klientom, w zależności od typu Service) komunikacje z podami aplikacji backendowej na porcie 4000, używajac protokołu TCP. Dzieki temu, niezależnie od tego, na których konkretnych podach działa aplikacja backendowa, ruch może być kierowany do nich przez stały punkt dostepu, którym jest ten Service. [2]

HPA backendu (HorizontalPodAutoscaler) automatycznie zwieksza lub zmniejsza liczbe podów w Deployment backend w zależności od obciażenia CPU, utrzymujac średnie wykorzystanie CPU na poziomie 50 procent. Dzieki temu aplikacja może elastycznie reagować na zmieniajace sie warunki obciażenia, zapewniajac odpowiednia wydajność przy optymalnym wykorzystaniu zasobów.

```
    apiVersion: apps/v1
kind: Deployment
metadata:
  name: backend
  namespace: myforum-namespace
spec:
  replicas: 5
  selector:
    matchLabels:
      app: backend
  template:
    metadata:
      labels:
        app: backend
    spec:
      containers:
        - name: backend
          image: pstrozyk644365/viteforum-backend:latest
          ports:
            - containerPort: 4000
          env:
            - name: MONGO_URL
              valueFrom:
                configMapKeyRef:
                  name: mongo-config
                  key: MONGO_URL
          resources:
            limits:
              memory: "512Mi"
              cpu: "500m"
            requests:
              memory: "256Mi"
              cpu: "250m"
          livenessProbe:
            httpGet:
              path: /health
              port: 4000
            initialDelaySeconds: 30
            periodSeconds: 10
          readinessProbe:
            httpGet:
              path: /health
              port: 4000
```

```
            initialDelaySeconds: 10
            periodSeconds: 5


----------------------
    apiVersion: v1
kind: Service
metadata:
  name: backend
  namespace: myforum-namespace
spec:
  selector:
    app: backend
  ports:
    - protocol: TCP
      port: 4000
      targetPort: 4000
```

1.1.2 Frontend

Deployment frontendu uruchamia jedna instancje aplikacji frontendowej dla forum, która jest dostepna na porcie 3000. Deployment monitoruje zdrowie aplikacji za pomoca sond i zarzadza zasobami, które aplikacja może zużywać.

Service frontendu w Kubernetes umożliwia dostep do aplikacji frontendowej z internetu na porcie 3000, używajac protokołu TCP. Dzieki typowi LoadBalancer, ruch z zewnatrz jest automatycznie przekierowywany do podów aplikacji, które pasuja do określonego selektora etykiet.

HorizontalPodAutoscaler (HPA) dla frontendu automatycznie skaluje liczbe podów w Deployment o nazwie frontend w przestrzeni nazw myforum-namespace, w zależności od wykorzystania CPU przez te pody.

```
    apiVersion: apps/v1
kind: Deployment
metadata:
  name: frontend
  namespace: myforum-namespace
spec:
  replicas: 5
  selector:
    matchLabels:
      app: frontend
  template:
    metadata:
      labels:
        app: frontend
    spec:
      containers:
        - name: frontend
```

```
          image: pstrozyk644365/viteforum-frontend:latest
          ports:
            - containerPort: 3000
          resources:
            limits:
              memory: "512Mi"
              cpu: "500m"
            requests:
              memory: "256Mi"
              cpu: "250m"
          livenessProbe:
            httpGet:
              path: /
              port: 3000
            initialDelaySeconds: 30
            periodSeconds: 10
          readinessProbe:
            httpGet:
              path: /
              port: 3000
            initialDelaySeconds: 10
            periodSeconds: 5
------------------------------

apiVersion: v1
kind: Service
metadata:
  name: frontend
  namespace: myforum-namespace
spec:
  selector:
    app: frontend
  ports:
    - protocol: TCP
      port: 3000
      targetPort: 3000
  type: LoadBalancer
```

1.1.3 Baza Danych

Deployment uruchamia MongoDB w pojedynczym pode w przestrzeni nazw myforum-namespace, eksponujac go na porcie 27017 i korzystajac z trwałego woluminu (Persistent Volume Claim) do przechowywania danych, co zapewnia trwałość danych bazy.

Service bazy danych umożliwia dostep do aplikacji MongoDB z innych podów w klastrze (lub z zewnatrz, w zależności od konfiguracji sieci i typu Service) na porcie 27017, używajac protokołu TCP.

PersistentVolumeClaim jest żadaniem o alokacje 1GB przestrzeni dyskowej, która może być podłaczona do jednego wezła na raz (tryb ReadWriteOnce) w przestrzeni nazw myforum-

namespace. PVC jest czesto używany do przechowywania danych w aplikacjach, które wymagaja trwałości danych poza cyklem życia podów, takich jak bazy danych. W tym przypadku, może być używany do przechowywania danych MongoDB.

```
    apiVersion: apps/v1
kind: Deployment
metadata:
  name: mongo
  namespace: myforum-namespace
spec:
  selector:
    matchLabels:
      app: mongo
  replicas: 1
  template:
    metadata:
      labels:
        app: mongo
    spec:
      containers:
        - name: mongo
          image: mongo:latest
          ports:
            - containerPort: 27017
          volumeMounts:
            - name: mongo-data
              mountPath: /data/db
      volumes:
        - name: mongo-data
          persistentVolumeClaim:
            claimName: mongo-pvc

---------------------------------------

    apiVersion: v1
kind: Service
metadata:
  name: mongo
  namespace: myforum-namespace
spec:
  selector:
    app: mongo
  ports:
    - protocol: TCP
      port: 27017
      targetPort: 27017

    apiVersion: v1
```

```
kind: PersistentVolumeClaim
metadata:
  name: mongo-pvc
  namespace: myforum-namespace
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
```

1.2 Opis infrastruktury - 6 pkt

Infrastruktura dla aplikacji jest zbudowana w oparciu o platforme Kubernetes z użyciem Minikube, która zarzadza wdrażaniem i skalowaniem aplikacji w klastrze. Aplikacja bedzie działać w wyizolowanym środowisku zwanym namespace o nazwie myforum-namespace. W ramach tego namespace zostały zdefiniowane i wdrożone nastepujace komponenty:

1. Deploymenty
    - Backend Deployment
        - Nazwa: backend
        - Liczba replik: Poczatkowo jedna, z możliwościa skalowania do maksymalnie pieciu replik dzieki konfiguracji Horizontal Pod Autoscaler.
        - Obraz Docker: pstrozyk644365/viteforum-backend:latest przechowywany w Docker Hub
        - Zasoby:
            * Limit pamieci: 512Mi
            * Limit CPU: 500m
            * Zasoby żadane: 256Mi pamieci, 250m CPU
        - Probes:
            * Liveness Probe: Sprawdza zdrowie podu co 10 sekund, po 30 sekundach od uruchomienia.
            * Readiness Probe: Sprawdza gotowość do obsługi żadań co 5 sekund, po 10 sekundach od uruchomienia.
            * Zasoby żadane: 256Mi pamieci, 250m CPU
        - Zmienne środowiskowe: URL do MongoDB pobierany z ConfigMap.
    - Frontend Deployment
        - Nazwa: frontend
        - Liczba replik: Jedna
        - Obraz Docker: pstrozyk644365/viteforum-frontend:latest
        - Zasoby:
            * Limit pamieci: 512Mi
            * Limit CPU: 500m

        * Zasoby żadane: 256Mi pamieci, 250m CPU
    - Probes:
        * Liveness Probe: Sprawdza zdrowie podu co 10 sekund, po 30 sekundach od uruchomienia.
        * Readiness Probe: Sprawdza gotowość do obsługi żadań co 5 sekund, po 10 sekundach od uruchomienia.
        * Zasoby żadane: 256Mi pamieci, 250m CPU
- MongoDB Deployment
    - Nazwa: mongo
    - Liczba replik: Jedna
    - Obraz Docker: mongo:latest
    - Pamieć masowa: Używa PersistentVolumeClaim (PVC) do przechowywania danych z dostepem ReadWriteOnce i rozmiarem 1Gi.

2. Horizontal Pod Autoscaler (HPA)
    - Nazwa: backend-autoscaler
    - Zakres replik: od 1 do 5
    - Metryka skalowania: Wykorzystanie CPU na poziomie 50 procent
3. ConfigMap
    - Nazwa: mongo-config
    - Dane: URL do MongoDB (MONGO_URL)
4. PersistentVolumeClaim (PVC)
    - Nazwa: mongo-pvc
    - Dostep: ReadWriteOnce
    - Rozmiar: 1Gi
5. Serwisy
    - Backend Service
        - Nazwa: backend
        - Typ: ClusterIP
        - Port: 4000
        - Selector: app: backend
    - Frontend Service
        - Nazwa: frontend
        - Typ: LoadBalancer
        - Port: 3000
        - Selector: app: frontend
    - MongoDB Service
        - Nazwa: mongo

- Typ: ClusterIP
- Port: 27017
- Selector: app: mongo

6. Wykorzystane narzedzia i platformy chmurowe

   Aplikacja wykorzystuje Kubernetes do orkiestracji kontenerów, zapewniajac wysoka dostepność, skalowalność i zarzadzanie zasobami. Podstawowe elementy infrastruktury obejmuja:

   Namespace: Izolacja środowiska aplikacji.
   Deployments: Zarzadzanie wdrażaniem kontenerów aplikacji.
   Services: Umożliwiaja komunikacje miedzy komponentami aplikacji oraz dostep zewnetrzny (w przypadku Frontendu).
   ConfigMap: Przechowywanie konfiguracji aplikacji.
   PersistentVolumeClaim: Zarzadzanie trwała pamiecia masowa dla MongoDB.

7. Zasoby sieciowe i pamieci masowej

   Sieci: Używane do komunikacji miedzy usługami i podami w klastrze Kubernetes.
   Pamieć masowa: PersistentVolumeClaim dla MongoDB zapewnia trwała pamieć masowa o rozmiarze 1Gi, umożliwiajac przechowywanie danych niezależnie od cyklu życia podów.

1.3 Opis komponentów aplikacji - 8 pkt

Backend obsługuje logike aplikacji oraz interakcje z baza danych. Jest zbudowany na obrazie Docker pstrozyk644365/viteforum-backend:latest. Uruchamia serwer HTTP z wykorzystaniem frameworka Express.js, który obsługuje różne żadania API dla aplikacji forum internetowego. Importuje niezbedne moduły takie jak Express, CORS (Cross-Origin Resource Sharing), Mongoose (do obsługi MongoDB), oraz modele użytkownika (User) i watku (Thread). Ustawia serwer Express, włacza obsługe danych formularza i JSON, oraz konfiguruje CORS. Nawiazuje połaczenie z baza danych MongoDB pod adresem mongodb://mongo:27017/viteForum. Skrypt uruchamia serwer forum internetowego, który umożliwia rejestracje i logowanie użytkowników, tworzenie watków i odpowiedzi, polubienie watków oraz pobieranie listy watków i ich odpowiedzi. Dane sa przechowywane w bazie danych MongoDB, a komunikacja z serwerem odbywa sie za pomoca żadań HTTP.

Frontend używa BrowserRouter z biblioteki react-router-dom do obsługi nawigacji i trasowania w aplikacji jednostronicowej (SPA). Wewnatrz BrowserRouter, Routes definiuje różne ścieżki (routes), które aplikacja może obsłużyć, i przypisuje do nich konkretne komponenty, które maja być renderowane, gdy użytkownik nawiguje do danej ścieżki. Oparty jest na vite-react, ponieważ Vite wykorzystuje natywne moduły ES do serwowania kodu, co znaczaco przyspiesza czas ładowania w trakcie rozwoju projektu. Dzieki temu zmiany w kodzie sa widoczne niemal natychmiastowo. Vite łatwo integruje sie z innymi narzedziami i bibliotekami używanymi w projektach React, takimi jak Redux, React Router czy styled-components, co sprawia, że jest wszechstronnym wyborem dla deweloperów.

MongoDB to nierelacyjna baza danych typu NoSQL, która przechowuje dane w formacie podobnym do JSON, znanym jako BSON (Binary JSON). Jest to rozwiazanie zaprojektowane z myśla o skalowalności, wydajności i łatwości użycia. MongoDB umożliwia

przechowywanie złożonych struktur danych, takich jak dokumenty, które moga zawierać różne typy danych i zagnieżdżone listy. Baza danych ta jest czesto wykorzystywana w aplikacjach internetowych i mobilnych ze wzgledu na swoja elastyczność i wydajność przy obsłudze dużych ilości danych oraz zapytań.
MongoDB jest wdrażany za pomoca Mongoose, biblioteki ODM (Object Data Modeling) dla MongoDB w środowisku Node.js Mongoose automatycznie obsługuje walidacje danych na podstawie zdefiniowanego schematu. Schematy i Modele umożliwiaja organizacje i zarzadzanie struktura danych aplikacji. Dzieki temu kod jest bardziej modularny i łatwiejszy w utrzymaniu.

1.4 Konfiguracja i zarzadzanie - 4 pkt

W Kubernetes Namespaces zapewniaja mechanizm izolacji grup zasobów w ramach pojedynczego klastra. Namespaces sa przeznaczone do użytku w środowiskach z wieloma użytkownikami rozproszonymi miedzy różnymi zespołami lub projektami. Namespaces definiuja zakres nazw. Nazwy zasobów musza być unikalne w obrebie jedngo namespace, ale nie musza być unikalne w obrebie globalnym. Namespaces nie moga być zagnieżdżane w sobie nawzajem, a każdy zasób Kubernetes może należeć tylko do jednej. Kubernetes [3]

Namespaces sa sposobem na podział zasobów klastra miedzy wielu użytkowników. Ten projekt na ten moment jest niewielki, a wiec został w nim użyty tylko jeden namespace myforum-namespace aby w przyszłości ułatwić izolacje zasobów aplikacji, co umożliwi łatwiejsze zarzadzanie i organizacje.

```
    apiVersion: v1
kind: Namespace
metadata:
  name: myforum-namespace
```

ConfigMap to obiekt API używany do przechowywania danych niepoufnych w postaci par klucz-wartość. Pody moga wykorzystywać ConfigMapy jako zmienne środowiskowe, argumenty wiersza poleceń lub pliki konfiguracyjne w woluminie.

ConfigMap umożliwia odseparowanie konfiguracji specyficznej dla środowiska od obrazów kontenerów, dzieki czemu aplikacje sa łatwo przenośne. ConfigMap w tej aplikacji przechowuje dane konfiguracyjne, takie jak URL do MongoDB, które sa używane przez backend.

```
    apiVersion: v1
kind: ConfigMap
metadata:
  name: mongo-config
  namespace: myforum-namespace
data:
  MONGO_URL: "mongodb://mongo:27017/viteForum"
```

HorizontalPodAutoscaler automatycznie aktualizuje zasoby obciażenia (takie jak Deployment lub StatefulSet), aby automatycznie skalować obciażenie w celu dopasowania go do zapotrzebowania.

Poziome skalowanie oznacza, że odpowiedzia na zwiekszone obciażenie jest uruchamianie dodatkowych Podów. Różni sie to od skalowania pionowego, które w kontekście Kubernetes oznaczałoby przypisanie wiekszej ilości zasobów (na przykład pamieci lub CPU) do już działajacych Podów dla danego obciażenia.

Jeśli obciażenie maleje, a liczba Podów jest powyżej skonfigurowanego minimum, HorizontalPodAutoscaler zleca zmniejszenie zasobów obciażenia (Deployment, StatefulSet lub inny podobny zasób).

PersistentVolumeClaim (PVC) to żadanie zasobów magazynowych przez użytkownika. Jest podobne do Podów. Pody zużywaja zasoby wezłów, a PVCs zużywaja zasoby PV (PersistentVolume).

Pody moga żadać określonych poziomów zasobów (CPU i pamieć). Z kolei PVC może żadać określonej wielkości i trybów dostepu (na przykład moga być montowane jako ReadWriteOnce, ReadOnlyMany, ReadWriteMany, lub ReadWriteOncePod). Dla tej aplikacji wykorzystane zostało ReadWriteOnce.

W Kubernetes, Service (Usługa) jest metoda ekspozycji aplikacji sieciowej, która działa jako jeden lub wiecej Podów w twoim klastrze.

Kluczowym celem Service w Kubernetes jest to, że nie musisz modyfikować istniejacej aplikacji, aby używać nieznanej mechanizmu odkrywania usług. Możesz uruchamiać kod w Podach, czy to kod zaprojektowany dla świata chmurowego, czy starsza aplikacje zkonteneryzowana. Używasz Services do udostepnienia tego zestawu Podów w sieci, aby klienci mogli z nimi interakcjonować.

1.5 Zarzadzanie błedami - 2 pkt

Probes (Liveness i Readiness): Konfiguracja probes (np. HTTP probe) do monitorowania zdrowia aplikacji. W przypadku awarii lub problemów z dostepnościa, Kubernetes może automatycznie restartować pod lub oznaczać je jako niegotowe.
Dodatkowo zostały użyte metrics do automatycznego skalowania aplikacji w razie przekroczenia pewnych zasobów.

```
    livenessProbe:
  httpGet:
    path: /health
    port: 4000
  initialDelaySeconds: 30
  periodSeconds: 10
readinessProbe:
  httpGet:
    path: /health
    port: 4000
  initialDelaySeconds: 10
  periodSeconds: 5
```

1.6 Skalowalność - 4 pkt

W projekcie został użyty mechanizm Horizontal Pod Autoscaler (HPA) w celu automatycznego skalowania liczby replik podów w zależności od obciażenia. Każdy z dwóch komponentów (frontend i backend) został skonfigurowany z osobnym HPA, który monitoruje zużycie zasobów CPU i pamieci, dostosowujac liczbe replik podów, aby utrzymać określony poziom wykorzystania zasobów.

Dla backendu HPA skaluje liczbe podów od 1 do 5, starajac sie utrzymać średnie wykorzystanie CPU na poziomie 50 procent. Podobnie, dla frontendu HPA również skaluje liczbe podów od 1 do 5, dażac do utrzymania średniego wykorzystania CPU na poziomie 50 procent.

Monitorowanie skalowania przez HPA odbywa sie w czasie rzeczywistym poprzez wykorzystanie wbudowanych metryk Kubernetes, które sa zbierane przez Kubernetes Metrics Server. Decyzje o skalowaniu sa podejmowane na podstawie tych metryk.

HPA jest podstawowym narzedziem do skalowania podów w Kubernetes, dynamicznie dostosowujacym liczbe replik w zależności od obciażenia, które jest mierzone przez metryki takie jak CPU i pamieć.

```
    apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: backend-autoscaler
  namespace: myforum-namespace
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: backend
  minReplicas: 1
  maxReplicas: 10
  metrics:
    - type: Resource
      resource:
        name: cpu
        target:
          type: Utilization
          averageUtilization: 50

    apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: frontend-hpa
  namespace: myforum-namespace
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
```

```
    name: frontend
  minReplicas: 1
  maxReplicas: 10
  metrics:
    - type: Resource
      resource:
        name: cpu
        target:
          type: Utilization
          averageUtilization: 50
```

1.7 Wymagania dotyczace zasobów - 2 pkt

1. Backend
   - CPU: Minimum 250 millicores (0.25 vCPU), limit 500 millicores (0.5 vCPU).
   - Pamieć RAM: Minimum 256 MiB, limit 512 MiB.
   - Miejsce na dysku: Backend nie wymaga dużego miejsca na dysku, ponieważ głównie działa jako aplikacja serwerowa obsługujaca zapytania.
   - Czas odpowiedzi: Backend powinien szybko odpowiadać na zapytania HTTP, z odpowiednim czasem odpowiedzi poniżej 1 sekundy w normalnych warunkach.
   - Odporność na obciażenie: Dzieki użyciu Horizontal Pod Autoscaler (HPA), backend powinien dynamicznie skalować sie w zależności od obciażenia, aby utrzymać stabilność i wydajność aplikacji
2. Frontend
   - CPU: Minimum 250 millicores (0.25 vCPU), limit 500 millicores (0.5 vCPU).
   - Pamieć RAM: Minimum 256 MiB, limit 512 MiB.
   - Miejsce na dysku: Frontend wymaga miejsca na przechowywanie kodu źródłowego i zasobów aplikacji webowej, co może zajać około 100 MB.
   - Czas odpowiedzi: Frontend powinien szybko renderować interfejs użytkownika w przegladarce, z minimalnym czasem ładowania strony dla użytkowników.
   - Odporność na obciażenie: Dzieki automatycznemu skalowaniu w Kubernetes, frontend powinien być w stanie obsługiwać rosnaca liczbe użytkowników bez utraty wydajności.
3. Database
   - CPU: Minimalne wymagania CPU dla MongoDB sa niskie, ale moga być zależne od ilości zapytań i operacji na bazie danych.
   - Pamieć RAM: Minimalne wymagania RAM zależa od ilości danych przechowywanych i wydajności operacji zapisu/odczytu.
   - Miejsce na dysku: MongoDB wymaga przestrzeni dyskowej na przechowywanie danych, zgodnie z ustawionym limitem w PersistentVolumeClaim: 1GB.
   - Odpowiedź na zapytania: MongoDB powinien szybko odpowiadać na zapytania bazodanowe, z minimalnym czasem oczekiwania.

- Stabilność i niezawodność: MongoDB wraz z PVC zapewniaja stabilność danych, nawet w przypadku awarii pojedynczych podów.

1.8 Architektura sieciowa - 4 pkt

Konfiguracja NetworkPolicy w Kubernetes, o nazwie allow-frontend-to-backend-and-backend-to-mongo, definiuje zasady bezpieczeństwa dla komunikacji miedzy aplikacjami w klastrze. NetworkPolicy obejmuje pody oznaczone jako app: backend i określa, że maja one zezwolenie na przyjmowanie ruchu z pody app: frontend na porcie TCP 4000 (Ingress). Dodatkowo, pody backend moga wysyłać ruch do pody app: frontend na porcie TCP 3000 oraz do pody app: mongo na porcie TCP 27017 (Egress). Ta konfiguracja precyzyjnie kontroluje komunikacje miedzy podami na podstawie ich etykiet, co jest kluczowe dla bezpieczeństwa i zarzadzania siecia w środowisku Kubernetes.

```
    apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-frontend-to-backend-and-backend-to-mongo
  namespace: myforum-namespace
spec:
  podSelector:
    matchLabels:
      app: backend
  policyTypes:
  - Ingress
  - Egress
  ingress:
  - from:
    - podSelector:
        matchLabels:
          app: frontend
    ports:
    - protocol: TCP
      port: 4000
  egress:
  - to:
    - podSelector:
        matchLabels:
          app: frontend
    ports:
    - protocol: TCP
      port: 3000
  - to:
    - podSelector:
        matchLabels:
          app: mongo
    ports:
    - protocol: TCP
```

```
port: 27017
```

Literatura

[1] AWS.docs, *What is a kubernetes cluster?*, 2022.

[2] Kubernetes.docs, *Service*, 2019.

[3] Nigel Poulton, *The kubernetes book*, LeanPub, 2021.