



Uniwersytet Gdański
Wydział Matematyki, Fizyki i Informatyki
Instytut Informatyki

Aplikacja do zarządzania zadaniami

Gracjan Kościński

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

Gdańsk
24 czerwca 2024

Spis treści

1	Opis projektu	2
1.1	Opis architektury	2
1.2	Opis infrastruktury	2
1.3	Opis komponentów aplikacji	3
1.4	Konfiguracja i zarządzanie	4
1.5	Zarządzanie błędami	5
1.6	Skalowalność	5
1.7	Wymagania dotyczące zasobów	5
1.8	Architektura sieciowa	6

1 Opis projektu

W dzisiejszym dynamicznie zmieniającym się świecie, firmy coraz częściej stają przed wyzwaniem skutecznego zarządzania swoimi aplikacjami i danymi. Firma ABC, napotkała problem zarządzania zadaniami przydzielanymi pracownikom oraz konieczności utrzymania wysokiej wydajności aplikacji przy jednoczesnym zapewnieniu skalowalności i bezpieczeństwa. Projekt ten ma na celu stworzenie rozwiązania opartego na Kubernetes, które integruje frontend, backend, bazę danych oraz system zarządzania tożsamościami Keycloak, aby sprostać tym wymaganiom.

1.1 Opis architektury

Architektura aplikacji jest oparta na Kubernetes, który zapewnia skalowalność, automatyzację zarządzania kontenerami oraz wysoką dostępność. Klaster Kubernetes zarządza wdrożeniami, skalowaniem oraz konfiguracją wszystkich komponentów aplikacji.

Komponenty architektury:

- **Kubernetes Cluster** Podstawowy element architektury, odpowiedzialny za zarządzanie kontenerami. Składa się z węzłów master i worker, które współpracują, aby zarządzać obciążeniem aplikacji.
- **Frontend** Aplikacja oparta na Next.js, uruchamiana w kontenerze. Obsługuje interfejs użytkownika i komunikuje się z backendem.
- **Backend** Serwis w python, który obsługuje logikę biznesową i komunikację z bazą danych oraz systemem Keycloak.
- **Baza danych (PostgreSQL)** Przechowuje dane aplikacji. Uruchamiana jako osobny kontener, z wykorzystaniem Persistent Volume Claim (PVC) do zapewnienia trwałości danych.
- **Keycloak** System zarządzania tożsamościami i uwierzytelnianiem. Zapewnia bezpieczne logowanie użytkowników oraz zarządzanie sesjami.

1.2 Opis infrastruktury

Aplikacja działa w środowisku chmurowym, wykorzystując platformę Kubernetes do zarządzania kontenerami. Infrastruktura składa się z następujących elementów:

- **Platforma chmurowa** Kubernetes
- **Sieć** Wykorzystanie ClusterIP do wewnętrznej komunikacji między usługami, oraz NodePort/LoadBalancer do eksponowania usług na zewnątrz klastra.
- **Pamięć masowa** Persistent Volume Claims (PVC) do zarządzania trwałością danych w bazie PostgreSQL.

1.3 Opis komponentów aplikacji

Każdy komponent aplikacji jest uruchamiany jako osobny kontener w klastrze Kubernetes.

1. Frontend

- **Implementacja** Next.js
- **Wdrożenie** Kontener Docker
- **Konfiguracja** Zmienne środowiskowe zarządzane przez ConfigMap
- **Zarządzanie** Automatyczne skalowanie

2. Backend

- **Implementacja** Python Flask
- **Wdrożenie** Kontener Docker
- **Konfiguracja** Zmienne środowiskowe zarządzane przez ConfigMap
- **Zarządzanie** Automatyczne skalowanie

3. PostgreSQL

- **Implementacja** PostgreSQL
- **Wdrożenie** Kontener Docker
- **Konfiguracja** Zmienne środowiskowe zarządzane przez ConfigMap i Secret
- **Zarządzanie** Persistent Volume Claim (PVC) zapewniające trwałość danych

4. Keycloak

- **Implementacja** Keycloak
- **Wdrożenie** Kontener Docker
- **Konfiguracja** Zmienne środowiskowe zarządzane przez ConfigMap
- **Zarządzanie** Automatyczne skalowanie

1.4 Konfiguracja i zarządzanie

Konfiguracja i zarządzanie aplikacją na poziomie klastra Kubernetes obejmuje:

- **ConfigMap** Przechowywanie konfiguracji niezawierającej wrażliwych danych
- **Secrets** Przechowywanie wrażliwych danych, takich jak hasła i klucze API.

Zarządzanie aplikacją w klastrze Kubernetes odbywa się również za pomocą Kubernetes Dashboard [1], które jest intuicyjnym interfejsem graficznym umożliwiającym monitorowanie i zarządzanie zasobami Kubernetes. Dashboard pozwala użytkownikom na przeglądanie stanu klastra, wdrożonych aplikacji oraz dostępnych zasobów w sposób bardziej przyjazny niż korzystanie wyłącznie z interfejsu wiersza poleceń `kubectl`.

Dzięki Kubernetes Dashboard, użytkownicy mogą:

- Monitorować ogólny stan klastra, w tym sprawność węzłów oraz działających na nich podów.
- Przeglądać szczegółowe informacje o wdrożonych aplikacjach, takich jak liczba replik, użycie zasobów, stan poszczególnych podów oraz logi aplikacji.
- Zarządzać konfiguracją aplikacji, w tym modyfikować ConfigMap i Secrets bezpośrednio z poziomu interfejsu graficznego.
- Tworzyć i usuwać zasoby, takie jak pody, usługi, deploymenty oraz inne obiekty Kubernetes.
- Sprawdzać aktualne użycie zasobów (CPU, pamięć) przez poszczególne komponenty aplikacji, co umożliwia optymalizację wykorzystania zasobów i szybszą identyfikację potencjalnych problemów wydajnościowych.

Aby dane dotyczące użycia zasobów były dostępne w Dashboard, konieczna jest instalacja i prawidłowa konfiguracja Metrics Server. Metrics Server jest kluczowym komponentem w ekosystemie Kubernetes, który zbiera metryki dotyczące użycia zasobów (CPU i pamięci) z kubeletów na węzłach i udostępnia je innym komponentom, takim jak Horizontal Pod Autoscaler, który automatycznie skalibruje liczbę replik podów w odpowiedzi na zmieniające się obciążenie aplikacji.

Metrics Server jest lekki i zaprojektowany do pracy w klastrach produkcyjnych. Gdy jest zainstalowany, Dashboard może wyświetlać aktualne metryki, co znacznie ułatwia monitorowanie wydajności i zdrowia aplikacji oraz szybką reakcję na wszelkie problemy. Dzięki temu administratorzy i deweloperzy mają lepszy wgląd w działanie swoich aplikacji i mogą skuteczniej zarządzać zasobami klastra.

Podsumowując, Kubernetes Dashboard w połączeniu z Metrics Server stanowi potężne narzędzie do zarządzania i monitorowania aplikacji w klastrze Kubernetes, zapewniając łatwość obsługi oraz głęboki wgląd w wydajność i stan działania wdrożonych aplikacji.

1.5 Zarządzanie błędami

Zarządzanie błędami w aplikacjach opartych na Kubernetes jest kluczowym elementem zapewnienia stabilności, niezawodności i wysokiej dostępności systemu. Skuteczne zarządzanie błędami obejmuje zarówno prewencję, jak i odpowiednie reakcje na występujące awarie, mające na celu szybkie przywrócenie działania aplikacji oraz minimalizację wpływu na użytkowników końcowych. **Sposoby zarządzania błędami**

- **Monitorowanie** Monitorowanie stanu aplikacji odgrywa kluczową rolę w wczesnym wykrywaniu potencjalnych problemów i błędów.
- **Automatyczne skalowanie i replikacja** W przypadku nagłego wzrostu obciążenia lub awarii jednego z podów, Kubernetes umożliwia automatyczne skalowanie i replikację podów za pomocą Horizontal Pod Autoscaler (HPA). HPA monitoruje metryki wydajnościowe i automatycznie dostosowuje liczbę replik podów w zależności od obciążenia aplikacji. Dzięki temu można zapewnić ciągłość działania aplikacji nawet w przypadku nieoczekiwanych wzrostów ruchu.

1.6 Skalowalność

Skalowalność jest jednym z fundamentalnych aspektów architektury aplikacji opartej na Kubernetes. W dynamicznie zmieniającym się środowisku, gdzie obciążenie aplikacji może gwałtownie wzrastać lub maleć, zapewnienie elastycznego i efektywnego skalowania jest nieodzownym elementem utrzymania wydajności oraz dostępności systemu.

Skalowanie w Kubernetes może być realizowane zarówno w pionie (skalowanie zasobów pojedynczego podu), jak i w poziomie (skalowanie liczby podów). W zależności od specyfiki aplikacji oraz wymagań biznesowych, można zastosować różne strategie skalowania. **Poziome Skalowanie**

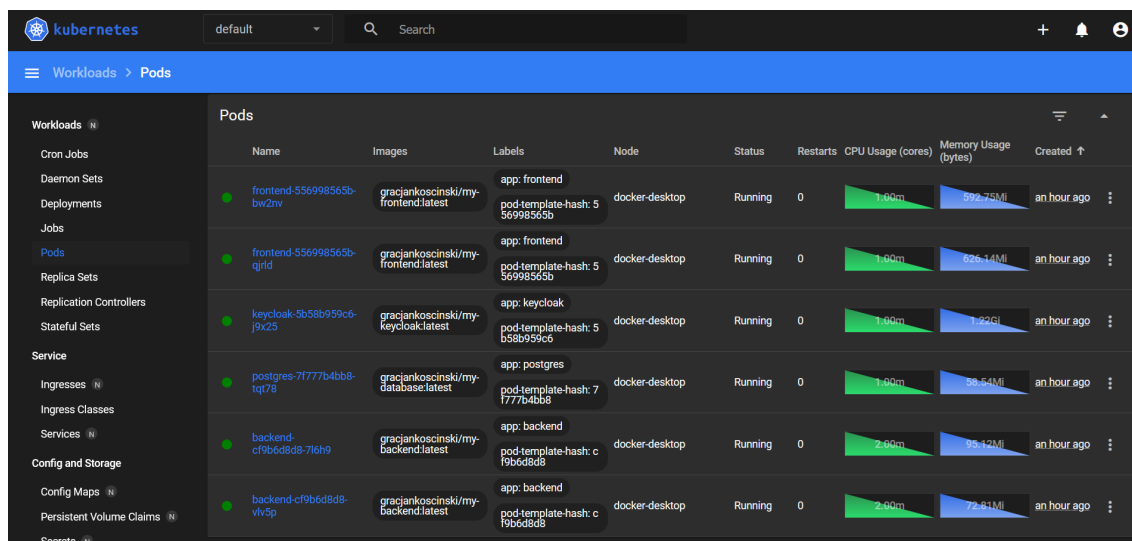
Poziome skalowanie, znane również jako skalowanie na zewnątrz (scaling out) i do wewnątrz (scaling in), polega na dodawaniu lub usuwaniu instancji podów w klastrze Kubernetes. Jednym z najważniejszych narzędzi wspomagających automatyczne poziome skalowanie jest Horizontal Pod Autoscaler (HPA) [2]. HPA monitoruje metryki zasobów, takie jak zużycie CPU czy pamięci, oraz dostosowuje liczbę replik podów w zależności od obciążenia. HPA konfiguruje się przy użyciu plików YAML, gdzie określa się minimalną i maksymalną liczbę replik oraz metryki, które mają być monitorowane.

1.7 Wymagania dotyczące zasobów

Każdy komponent aplikacji ma swoje specyficzne wymagania dotyczące zasobów, takie jak ilość pamięci RAM, CPU, miejsce na dysku itp. Aby zapewnić optymalne działanie aplikacji, zasoby te muszą być odpowiednio przydzielone. W naszym przypadku, dla komponentów takich jak frontend, backend, keycloak i postgres, wymagania dotyczące zasobów zostały szczegółowo określone na podstawie analizy zużycia zasobów.

Dzięki zastosowaniu Kubernetes Dashboard oraz Metrics Server, mogliśmy monitorować zużycie zasobów przez poszczególne komponenty aplikacji. Dashboard dostarcza szczegółowych informacji o aktualnym zużyciu CPU, pamięci RAM oraz innych zasobów

przez każdy pod w klastrze. Na podstawie tych danych mogliśmy precyzyjnie określić optymalne wartości dla requests i limits zasobów w plikach konfiguracyjnych.



Name	Images	Labels	Node	Status	Restarts	CPU Usage (cores)	Memory Usage (bytes)	Created
frontend-556998565b-bw2iv	gracjankoscinski/my-frontend:latest	app: frontend pod-template-hash: 556998565b	docker-desktop	Running	0	1.00m	392.75Mi	an hour ago
frontend-556998565b-qrid	gracjankoscinski/my-frontend:latest	app: frontend pod-template-hash: 556998565b	docker-desktop	Running	0	1.00m	626.14Mi	an hour ago
keycloak-5b58b959c6-j9x25	gracjankoscinski/my-keycloak:latest	app: keycloak pod-template-hash: 5b58b959c6	docker-desktop	Running	0	1.00m	1.42Gi	an hour ago
postgres-7f777b4bb8-tqt78	gracjankoscinski/my-database:latest	app: postgres pod-template-hash: 7f777b4bb8	docker-desktop	Running	0	1.00m	56.64Mi	an hour ago
backend-cf9bd8d8-7f6h9	gracjankoscinski/my-backend:latest	app: backend pod-template-hash: cf9bd8d8	docker-desktop	Running	0	2.00m	95.12Mi	an hour ago
backend-cf9bd8d8-vlv5p	gracjankoscinski/my-backend:latest	app: backend pod-template-hash: cf9bd8d8	docker-desktop	Running	0	2.00m	72.81Mi	an hour ago

Rysunek 1: Zużycie zasobów.



Rysunek 2: Zużycie zasobów.

1.8 Architektura sieciowa

Architektura sieciowa [3] aplikacji w kontekście środowiska opartego na Kubernetes pełni kluczową rolę w zapewnieniu prawidłowego funkcjonowania oraz bezpieczeństwa komunikacji między komponentami aplikacji oraz z zewnętrznymi użytkownikami. Zastosowanie odpowiednich mechanizmów komunikacyjnych jest niezbędne do skutecznego zarządzania ruchem sieciowym oraz zapewnienia wysokiej dostępności i wydajności systemu.

- **ClusterIP Services** Usługi typu ClusterIP są używane do wewnętrznej komunikacji między różnymi komponentami aplikacji w obrębie klastra Kubernetes. Każda usługa typu ClusterIP otrzymuje wirtualny adres IP, który jest dostępny tylko w obrębie klastra. To pozwala na bezpieczną i efektywną komunikację między aplikacjami, które mogą być uruchomione na różnych węzłach w klastrze. Usługi typu ClusterIP są zazwyczaj wykorzystywane do zapewnienia komunikacji między mikroservisami oraz komponentami wewnętrznymi aplikacji.
- **NodePort/LoadBalancer Services** Usługi typu NodePort umożliwiają eksponowanie aplikacji na zewnątrz klastra Kubernetes poprzez przypisanie stałego portu

na każdym węźle klastra. To pozwala na dostęp do aplikacji z poziomu zewnętrznych systemów oraz urządzeń. NodePort jest używany w przypadkach, gdy aplikacja musi być dostępna z zewnątrz klastra, np. do interakcji z klientami, użytkownikami lub innymi systemami zewnętrznymi.

Literatura

- [1] *Deploy and access the kubernetes dashboard*, 2024.
- [2] *Horizontal pod autoscaling*, 2024.
- [3] *Services and networking in kubernetes*, 2024.