Uniwersytet Gdański
Wydział Matematyki, Fizyki i Informatyki
Instytut Informatyki

Blogium

Stanisław Olszewski

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

Gdańsk

26 czerwca 2024

Spis treści

1 Opis projektu

Blogium to wyjątkowe miejsce, gdzie użytkownicy mogą dzielić się swoimi przeżyciami i inspiracjami. Platforma dostępna jest wyłącznie dla zalogowanych użytkowników, dlatego warto jak najszybciej założyć konto. Po zalogowaniu, otwierają się przed nami pełne możliwości aplikacji. Możemy zaryzykować i odwiedzić losowy blog, odkrywając nowe historie, lub skorzystać z wyszukiwarki, aby znaleźć konkretnego autora i cieszyć się lekturą jego przygód.

Kiedy przeczytamy już wszystko, co nas interesuje, możemy stworzyć własnego bloga. Na nim to my będziemy jedynymi autorami wpisów, co daje pełną kontrolę nad treściami. Jedynie administrator ma prawo usunąć naszego bloga, co zapewnia porządek i bezpieczeństwo na platformie. Blogium to miejsce, gdzie każdy może znaleźć coś dla siebie i dzielić się swoimi opowieściami z innymi.

1.1 Opis architektury - 8 pkt

Aplikacja Blogium składa się z pięciu kluczowych komponentów: interfejsu użytkownika (NextJS), serwera aplikacji (Python FastAPI), bazy danych MongoDB, bazy danych PostgreSQL oraz systemu zarządzania tożsamością Keycloak.

Blogium działa w klastrze Kubernetes, który jest open-source'owym rozwiązaniem do zarządzania kontenerami. Kubernetes zapewnia wysoką skalowalność, niezawodność i wydajność aplikacji.

Użytkownicy mają dostęp do frontendu za pośrednictwem przeglądarki internetowej pod adresem localhost:3000 dzięki port-forwardingowi. Serwer aplikacji komunikuje się z bazą danych MongoDB, gdzie przechowywane są blogi użytkowników. MongoDB posiada swój własny wolumen, który zapewnia przechowywanie danych nawet w przypadku awarii. Zarządzanie dostępem do wszystkich komponentów aplikacji odbywa się za pomocą Keycloak, który swoje dane przechowuje w bazie PostgreSQL.

Dzięki tej architekturze, Blogium jest zarówno wydajne, jak i niezawodne, a także łatwe do skalowania w miarę wzrostu liczby użytkowników i zawartości.

1.2 Opis infrastruktury - 6 pkt

Serwis Blogium został zaprojektowany do działania w klastrze Kubernetes w wersji 1.29.2, uruchomionym za pomocą narzędzia Docker Desktop. To narzędzie pozwala na uruchamianie kontenerów i klastrów Kubernetes bezpośrednio na lokalnym komputerze, co stanowi idealne rozwiązanie do testowania i tworzenia aplikacji bez potrzeby konfigurowania pełnowymiarowego klastra.

Domyślna specyfikacja klastra to 4 GB pamięci RAM, 2 wirtualne CPU oraz 10 GB przestrzeni dyskowej, co zapewnia wystarczające zasoby do efektywnego testowania i rozwijania serwisu Blogium w środowisku lokalnym.

1.3 Opis komponentów aplikacji - 8 pkt

Powinna zawierać informacje na temat komponentów aplikacji, takich jak serwisy, aplikacje i bazy danych. W szczególności należy zwrócić uwagę na sposoby ich wdrażania, konfiguracji i zarządzania.

Serwis Blogium składa się z następujących komponentów:

- **Frontend:** Interfejs internetowy dla użytkownika, stworzony przy użyciu frameworka NextJS, który rozszerza bibliotekę React. NextJS wprowadza wiele zaawansowanych funkcjonalności, takich jak akcje serwerowe i AuthJS.
- **Backend:** Serwer aplikacji zbudowany z użyciem FastAPI w Pythonie. Komunikuje się z bazą danych MongoDB za pomocą biblioteki PyMongo oraz dekoduje tokeny JWT, zapewniając bezpieczeństwo i autoryzację użytkowników.
- **MongoDB:** Nierelacyjna baza danych zaprojektowana do przechowywania danych w formie dokumentów. Przechowuje blogi użytkowników na dedykowanym wolumenie, co zapewnia dodatkową niezawodność i ochronę danych.
- **Keycloak:** System zarządzania użytkownikami, skonfigurowany do autentykacji przy użyciu protokołu OpenID Connect. Generuje tokeny użytkowników, które określają poziom dostępu do aplikacji.
- **PostgreSQL:** Baza danych używana do przechowywania danych Keycloak'a. Podobnie jak MongoDB, posiada dedykowany wolumen, zapewniając bezpieczeństwo i trwałość danych.

Dzięki tej architekturze, Blogium oferuje kompleksowe rozwiązanie do zarządzania blogami, łącząc nowoczesne technologie frontendowe i backendowe z niezawodnym przechowywaniem danych i zaawansowanym systemem zarządzania użytkownikami.

1.4 Konfiguracja i zarządzanie - 4 pkt

Powinna zawierać informacje na temat konfiguracji i zarządzania aplikacją na poziomie klastra Kubernetes.

Konfiguracja i zarządzanie klastrem Kubernetes dla serwisu Blogium odbywa się przy użyciu następujących zasobów:

- **Deployments:** Każdy komponent aplikacji, z wyjątkiem baz danych, jest uruchamiany jako oddzielny Deployment. Deployments umożliwiają łatwe skalowanie oraz aktualizację parametrów kontenerów, co pozwala na efektywne zarządzanie aplikacją.
- **Services:** Każdy Deployment ma przypisany własny Service, który umożliwia komunikację między kontenerami oraz równoważenie obciążenia, zapewniając stabilne działanie aplikacji.
- **StatefulSet:** Bazy danych są uruchamiane jako StatefulSet. Każda baza danych posiada unikalną tożsamość, która jest zachowywana podczas restartów i migracji na inne węzły, co zapewnia trwałość i niezawodność przechowywanych danych.

- **ConfigMaps i Secrets:** ConfigMaps służą do przechowywania jawnych danych konfiguracyjnych, takich jak konfiguracja Keycloak. Secrets natomiast przechowują poufne informacje zaszyfrowane w formacie BASE64. Obie te struktury umożliwiają szybkie i bezpieczne zarządzanie parametrami aplikacji bez konieczności jej ponownego kompilowania.
- **Persistent Volume Claims (PVC):** PVC to wolumeny, które zapewniają trwałe przechowywanie danych nawet w przypadku usunięcia lub awarii kontenera. Dzięki nim dane są chronione i mogą być odzyskane po awarii.

Dzięki tym zasobom i mechanizmom, klaster Kubernetes zapewnia efektywne i bezpieczne zarządzanie serwisem Blogium, umożliwiając jego skalowanie, aktualizację oraz trwałe przechowywanie danych.

1.5 Zarządzanie błędami - 2 pkt

Błędy w serwisie Blogium są kontrolowane w następujący sposób:

- **Na poziomie aplikacji:**
  - **Frontend:** Fragmenty kodu potencjalnie mogące generować błędy są opatrzone blokami "try-catch". Zapytania do serwera, które kończą się niepowodzeniem, wyświetlają użytkownikowi odpowiedni komunikat, zapewniając przejrzystość i zrozumienie sytuacji.
  - **Backend:** Analogicznie, po stronie backendu używane są bloki "try-except", które w przypadku błędu wysyłają odpowiedni komunikat i status do klienta, informując, co poszło nie tak.
- **Na poziomie klastra:** Kubernetes zapewnia niemal natychmiastową replikację poda w przypadku awarii, co minimalizuje przestoje w działaniu serwisu. Jeśli pod ma więcej niż jedną replikę, ruch jest przekierowywany do pozostałych replik, podczas gdy uszkodzony pod się restartuje. Dzięki temu użytkownicy serwisu nie odczuwają praktycznie żadnych przerw w działaniu.

1.6 Skalowalność - 4 pkt

Skalowalność jest fundamentem architektury aplikacji opartej na Kubernetes. Wyróżniamy dwa główne rodzaje skalowalności: horyzontalną i wertykalną.

- **Skalowalność Horyzontalna:** Polega na dodawaniu lub usuwaniu instancji (podów) aplikacji, aby dostosować się do zmieniającego się obciążenia. W Kubernetes realizuje się to głównie za pomocą ReplicaSet (ręczne ustawienie liczby replik Deploymentu) lub Horizontal Pod Autoscaler (automatyczne dostosowywanie liczby podów na podstawie metryk, takich jak CPU, pamięć, czy niestandardowe metryki użytkownika). Skalowanie horyzontalne umożliwia łatwe dostosowanie liczby instancji aplikacji do bieżącego zapotrzebowania, zapewniając elastyczność i wysoką dostępność.

- **Skalowalność Wertykalna:** Polega na dostosowywaniu zasobów przydzielonych pojedynczym instancjom (podom). W Kubernetes realizuje się to za pomocą Vertical Pod Autoscaler (VPA), który automatycznie dostosowuje limity i żądania zasobów (CPU i pamięci) dla podów. VPA monitoruje zasoby używane przez pody i dynamicznie je dostosowuje, aby zoptymalizować wydajność aplikacji. Skalowanie wertykalne pozwala na efektywne wykorzystanie dostępnych zasobów, minimalizując marnotrawstwo i zapewniając stabilność aplikacji.

Dzięki zastosowaniu tych dwóch rodzajów skalowalności, Kubernetes umożliwia elastyczne i efektywne zarządzanie zasobami, dostosowując się do zmieniających się warunków i wymagań aplikacji.

1.7 Wymagania dotyczące zasobów - 2 pkt

- **Frontend:** 512MB pamięci RAM, 0.4 rdzenia CPU
- **Backend:** 512MB pamięci RAM, 0.3 rdzenia CPU
- **Keycloak:** 1.5GB pamięci RAM, 1 rdzenia CPU
- **MongoDB:** 1GB pamięci na dysku
- **PostgreSQL:** 1GB pamięci na dysku

Zasoby podane dla każdego komponentu powinny zapewnić wystarczającą wydajność i czas odpowiedzi.

1.8 Architektura sieciowa - 4 pkt

Architektura sieciowa aplikacji opiera się na wykorzystaniu service'ów, które umożliwiają komunikację między poszczególnymi komponentami. Użytkownicy uzyskują dostęp do aplikacji poprzez port-forwarding, co przekierowuje porty 3000, 8000 i 8080 z serwera klastra na odpowiednie porty lokalne. Frontend oraz backend komunikują się ze sobą za pomocą protokołu HTTP, natomiast autentykacja użytkowników jest obsługiwana poprzez protokół OpenID Connect we współpracy z serwerem Keycloak.

Literatura

[1] Harsh Bhandari, *Implementing authentication in next.js*, 2024.

[2] Benjamin Buffet, *Securing fastapi with keycloak*, 2024.

[3] Shubham Dhote, *Keycloak deployment on kubernetes cluster*, 2021.

[4] Ulaş Özdemir, *Enterprise-level authentication in a containerized environment*, 2024.