Uniwersytet Gdański
Wydział Matematyki, Fizyki i Informatyki
Instytut Informatyki

Weather app

Oskar Gawryszewski

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

Gdańsk

26 czerwca 2024

Spis treści

1 Opis projektu

Projekt "Weather app"to aplikacja sieciowa, umożliwiajaca sprawdzanie warunków pogodowych na całym świecie. Pierwsza wersja projektu powstała na rzecz rekrutacji do firmy "Nokia". Pierwsza wersja składała sie z samego frontendu, który później został zdeployowany na AWS. Aktualna wersja "Weather app"to rozszerzona wersja tamtego projektu. Do aplikacji dodałem baze danych, przechowujaca historie wyszukiwania pogody, autoryzacje i autentykacje używajac Keycloak, oraz panel admina do zarzadzania zasobami.

1.1 Opis architektury - 8 pkt

Architektura aplikacji oparta na Kubernetes składa sie z kilku kluczowych komponentów, które zapewniaja skalowalność, wysoka dostepność i zarzadzanie kontenerami. Wykorzystane elementy Kubernetes obejmuja Deployment, Service, Ingress oraz Keycloak do zarzadzania uwierzytelnianiem.

1.1.1 Deployment

Deployment jest zasobem Kubernetes, który zapewnia deklaratywne zarzadzanie aplikacjami. Definiuje on, jakie obrazy kontenerów maja być uruchomione, ile replik ma być utrzymywanych oraz jakie zmienne środowiskowe maja być ustawione. W projekcie wykorzystano nastepujace Deploymenty:

- **server-deployment**: Zarzadza instancjami backendu aplikacji, zapewniajac skalowalność i niezawodność poprzez utrzymywanie trzech replik serwera.
- **redis-deployment**: Zarzadza instancja Redis, która jest wykorzystywana jako baza danych pamieci podrecznej, zapewniajac wysoka wydajność operacji odczytu/zapisu.
- **client-deployment**: Zarzadza instancjami frontendowej cześci aplikacji, obsługujac żadania użytkowników końcowych.

1.1.2 Service

Service jest zasobem Kubernetes, który definiuje sposób komunikacji miedzy różnymi podami oraz zapewnia stabilny adres IP i DNS dla zestawu podów. W projekcie użyto nastepujacych Service:

- **server-cluster-ip-service**: Udostepnia stabilny adres IP dla backendu, umożliwiajac innym komponentom aplikacji komunikacje z serwerem.
- **redis-cluster-ip-service**: Udostepnia stabilny adres IP dla usługi Redis, umożliwiajac backendowi dostep do bazy danych pamieci podrecznej.
- **client-cluster-ip-service**: Udostepnia stabilny adres IP dla frontendowej cześci aplikacji, umożliwiajac użytkownikom końcowym dostep do aplikacji.

1.1.3 Ingress

Ingress jest zasobem Kubernetes, który zarzadza dostepem zewnetrznym do usług w klastrze, zazwyczaj poprzez HTTP. W projekcie wykorzystano Ingress do zarzadzania ruchem do aplikacji:

- **ingress**: Konfiguruje reguły routingu, aby skierować ruch HTTP/HTTPS do odpowiednich usług, takich jak backend czy frontend aplikacji, umożliwiajac dostep z zewnatrz klastra.

1.1.4 Keycloak

Keycloak jest narzedziem do zarzadzania tożsamościami i dostepem, które zapewnia funkcje takie jak logowanie jednokrotne (SSO), uwierzytelnianie i autoryzacja. W projekcie Keycloak jest uruchomiony jako osobny kontener Docker, który zarzadza uwierzytelnianiem użytkowników aplikacji.

1.1.5 Interakcja komponentów

Poszczególne komponenty aplikacji komunikuja sie ze soba za pośrednictwem zdefiniowanych usług (Service). Frontend wysyła żadania HTTP do backendu poprzez **server-cluster-ip-service**. Backend korzysta z usługi Redis za pośrednictwem **redis-cluster-ip-service** do operacji zwiazanych z pamiecia podreczna. Ingress zarzadza ruchem zewnetrznym, kierujac go do odpowiednich usług, co umożliwia użytkownikom dostep do aplikacji przez przegladarke internetowa. Wszystkie te komponenty współdziałaja, aby zapewnić kompleksowa, skalowalna i bezpieczna aplikacje działajaca w środowisku Kubernetes.

1.2 Opis infrastruktury - 6 pkt

Aplikacja została uruchomiona w środowisku Kubernetes, co umożliwia zarzadzanie kontenerami na duża skale oraz automatyzacje wdrożeń, skalowanie i zarzadzanie zasobami aplikacyjnymi. Poniżej przedstawiono szczegółowy opis infrastruktury, w której działa aplikacja, oraz wykorzystywane narzedzia i zasoby.

1.2.1 Środowisko uruchomieniowe

Aplikacja działa w klastrze Kubernetes, który jest zarzadzany lokalnie za pomoca narzedzia `Docker Desktop`. Docker Desktop umożliwia uruchamianie pojedynczego wezła klastra Kubernetes na lokalnej maszynie, co jest idealnym rozwiazaniem do testowania i rozwoju aplikacji. Kubernetes zapewnia zarzadzanie kontenerami, automatyzacje wdrożeń oraz monitorowanie stanu aplikacji.

1.2.2 Wykorzystane narzedzia

Do zarzadzania klastrem Kubernetes oraz wdrażania aplikacji wykorzystano nastepujace narzedzia:

- `kubectl` - Komenda CLI do zarzadzania zasobami Kubernetes.
- `Docker` - Platforma do tworzenia, wdrażania i uruchamiania aplikacji w kontenerach. Obrazy Docker dla aplikacji sa przechowywane w Docker Hub.
- `Nginx` - Serwer HTTP i reverse proxy używany do obsługi frontendu aplikacji oraz przekierowywania ruchu do backendu.

1.2.3 Komponenty infrastruktury

Aplikacja składa sie z kilku komponentów, które zostały wdrożone jako zasoby Kubernetes:

- **client-deployment.yaml** - Definiuje wdrożenie frontendu aplikacji, który jest serwowany przez serwer Nginx.
- **server-deployment.yaml** - Definiuje wdrożenie backendu aplikacji, który jest odpowiedzialny za przetwarzanie logiki aplikacji i komunikacje z baza danych Redis.
- **redis-deployment.yaml** - Definiuje wdrożenie serwera Redis, który jest używany jako baza danych do przechowywania tymczasowych danych aplikacji.
- **client-cluster-ip-service.yaml** - Konfiguruje usługe Kubernetes typu ClusterIP, która eksponuje frontend aplikacji wewnatrz klastra.
- **server-cluster-ip-service.yaml** - Konfiguruje usługe Kubernetes typu ClusterIP, która eksponuje backend aplikacji wewnatrz klastra.
- **redis-cluster-ip-service.yaml** - Konfiguruje usługe Kubernetes typu ClusterIP, która eksponuje serwer Redis wewnatrz klastra.
- **ingress.yaml** - Konfiguruje zasób Ingress, który zarzadza zewnetrznym dostepem do usług w klastrze Kubernetes, rozdzielajac ruch do odpowiednich usług na podstawie reguł URL.
- **ingress-class.yaml** - Definiuje klase Ingress używana przez kontroler Ingress Nginx.

1.2.4 Zarzadzanie zasobami

W celu efektywnego zarzadzania zasobami, takimi jak sieci i pamieć masowa, zastosowano nastepujace rozwiazania:

- **Sieci** - Wykorzystano wewnetrzna sieć Kubernetes do komunikacji miedzy różnymi komponentami aplikacji. Każda usługa jest eksponowana wewnatrz klastra za pomoca usługi typu ClusterIP, co umożliwia bezpieczna i wydajna komunikacje miedzy mikroserwisami.
- **Pamieć masowa** - Aplikacja wykorzystuje Redis jako baze danych do przechowywania tymczasowych danych. Redis jest wdrożony jako stateful set, co zapewnia trwałość danych nawet w przypadku restartu podów.

Podsumowujac, infrastruktura aplikacji została zaprojektowana w sposób umożliwiajacy łatwe zarzadzanie, skalowanie oraz automatyzacje procesów wdrożeniowych. Kubernetes, jako fundament architektury, zapewnia elastyczność i niezawodność, co jest kluczowe dla efektywnego zarzadzania nowoczesnymi aplikacjami kontenerowymi.

1.3 Opis komponentów aplikacji - 8 pkt

Aplikacja składa sie z kilku kluczowych komponentów, które działaja razem, aby zapewnić pełna funkcjonalność systemu. Komponenty te obejmuja frontend, backend oraz baze danych Redis. Poniżej przedstawiono szczegółowy opis każdego z tych komponentów, w tym sposoby ich wdrażania, konfiguracji i zarzadzania.

1.3.1 Frontend

Frontend aplikacji jest implementowany jako aplikacja React, która jest serwowana przez serwer Nginx. Komponent ten jest wdrażany za pomoca pliku `client-deployment.yaml`, który definiuje specyfikacje dla deploymentu Kubernetes. Usługa frontendu jest konfigurowana przez plik `client-cluster-ip-service.yaml`, który eksponuje frontend wewnatrz klastra Kubernetes.

1.3.2 Backend

Backend aplikacji jest odpowiedzialny za przetwarzanie logiki aplikacji i obsługe zapytań od frontendu. Jest wdrażany jako serwer Node.js za pomoca pliku `server-deployment.yaml`. Konfiguracja usługi backendu jest zdefiniowana w pliku `server-cluster-ip-service.yaml`, który eksponuje backend wewnatrz klastra Kubernetes.

1.3.3 Baza danych Redis

Redis jest wykorzystywany jako baza danych do przechowywania tymczasowych danych aplikacji. Redis jest wdrażany za pomoca pliku `redis-deployment.yaml`, a jego konfiguracja jako usługi jest zdefiniowana w pliku `redis-cluster-ip-service.yaml`. Redis jest kluczowy dla wydajności aplikacji, zapewniajac szybki dostep do danych.

1.3.4 Ingress Controller

Do zarzadzania zewnetrznym dostepem do aplikacji wykorzystywany jest Ingress Controller, który jest konfigurowany za pomoca plików `ingress-class.yaml` oraz `ingress.yaml`. Ingress Controller umożliwia zarzadzanie ruchem HTTP(S) do różnych usług w klastrze Kubernetes na podstawie reguł URL.

1.4 Konfiguracja i zarzadzanie - 4 pkt

Konfiguracja i zarzadzanie aplikacja na poziomie klastra Kubernetes odbywa sie za pomoca plików YAML, które definiuja deploymenty, usługi oraz zasoby Ingress. Każdy

komponent aplikacji ma swój własny plik konfiguracyjny, co umożliwia łatwe zarzadzanie i modyfikowanie poszczególnych cześci systemu.

1.4.1 Konfiguracja deploymentów

Deploymenty sa konfigurowane za pomoca plików YAML, które definiuja liczebność replik, obrazy Docker oraz zmienne środowiskowe. Na przykład, plik `server-deployment.yaml` definiuje deployment backendu z trzema replikami, które moga być skalowane w zależności od obciażenia.

1.4.2 Konfiguracja usług

Usługi sa konfigurowane za pomoca plików `*-cluster-ip-service.yaml`, które definiuja typ usługi (ClusterIP) oraz porty, na których usługi sa dostepne. Usługi te umożliwiaja komunikacje miedzy komponentami aplikacji wewnatrz klastra Kubernetes.

1.4.3 Zarzadzanie Ingress

Ingress jest zarzadzany za pomoca plików `ingress.yaml` oraz `ingress-class.yaml`, które definiuja reguły routingu dla ruchu HTTP(S). Ingress Controller Nginx obsługuje ruch przychodzacy i przekierowuje go do odpowiednich usług w klastrze na podstawie ścieżek URL.

1.5 Zarzadzanie błedami - 2 pkt

Zarzadzanie błedami w aplikacji odbywa sie poprzez monitorowanie logów i stanu podów w klastrze Kubernetes. Narzedzia takie jak `kubectl logs` oraz `kubectl describe pod` sa wykorzystywane do diagnostyki problemów i analizy błedów.

1.5.1 Monitorowanie stanu aplikacji

Stan aplikacji jest monitorowany za pomoca komend `kubectl get pods` oraz `kubectl get services`, które pokazuja aktualny status podów i usług. W przypadku wykrycia problemów, logi podów sa analizowane w celu identyfikacji przyczyny błedów.

1.5.2 Reagowanie na awarie

W przypadku awarii, Kubernetes automatycznie restaruje nieudane pody, co zapewnia wysoka dostepność aplikacji. Dodatkowo, deploymenty sa skonfigurowane tak, aby automatycznie skalować liczbe replik w odpowiedzi na zmiany obciażenia, co minimalizuje ryzyko awarii spowodowanych przeciażeniem.

1.6 Skalowalność - 4 pkt

Skalowalność jest kluczowym elementem architektury aplikacji opartej na Kubernetes. Aplikacja może być skalowana zarówno pionowo, poprzez zwiekszenie zasobów dostepnych dla poszczególnych podów, jak i poziomo, poprzez dodanie wiekszej liczby replik.

1.6.1 Automatyczne skalowanie

Kubernetes umożliwia automatyczne skalowanie aplikacji za pomoca Horizontal Pod Autoscaler (HPA), który monitoruje metryki takie jak CPU i pamieć, i na ich podstawie dostosowuje liczbe replik podów. HPA jest konfigurowany za pomoca plików YAML, które definiuja zasady skalowania.

1.6.2 Monitorowanie skalowania

Skalowanie jest monitorowane za pomoca narzedzi takich jak `kubectl top`, które pokazuja aktualne zużycie zasobów przez pody i usługi. Na podstawie tych danych administratorzy moga dostosowywać konfiguracje HPA oraz zasoby dostepne dla aplikacji.

1.7 Wymagania dotyczace zasobów - 2 pkt

Każdy komponent aplikacji ma określone wymagania dotyczace zasobów, które sa skonfigurowane w plikach YAML deploymentów. Wymagania te obejmuja zapotrzebowanie na pamieć RAM, CPU oraz miejsce na dysku. Poniżej przedstawiono szczegółowe wymagania dla poszczególnych komponentów aplikacji:

1.7.1 Frontend (client-deployment.yaml)

Frontend aplikacji wymaga nastepujacych zasobów:

- **Zapotrzebowanie na CPU:** 100m
- **Zapotrzebowanie na pamieć RAM:** 100Mi
- **Limit CPU:** 200m
- **Limit pamieci RAM:** 200Mi

1.7.2 Backend (server-deployment.yaml)

Backend aplikacji wymaga nastepujacych zasobów:

- **Zapotrzebowanie na CPU:** 100m
- **Zapotrzebowanie na pamieć RAM:** 100Mi
- **Limit CPU:** 200m
- **Limit pamieci RAM:** 200Mi

1.7.3 Baza danych Redis (redis-deployment.yaml)

Baza danych Redis wymaga nastepujacych zasobów:

- **Zapotrzebowanie na CPU:** 100m
- **Zapotrzebowanie na pamieć RAM:** 100Mi
- **Limit CPU:** 200m
- **Limit pamieci RAM:** 200Mi

1.8 Architektura sieciowa - 4 pkt

Architektura sieciowa aplikacji opiera sie na klastrze Kubernetes, który zarzadza komunikacja miedzy różnymi komponentami aplikacji. Każdy komponent (frontend, backend, baza danych) jest wdrażany jako osobny pod i komunikuje sie z innymi komponentami za pomoca usług Kubernetes.

1.8.1 Konfiguracja sieci w klastrze Kubernetes

W klastrze Kubernetes, każdy pod ma unikalny adres IP, a komunikacja miedzy podami jest realizowana za pomoca sieci wirtualnych. Usługi Kubernetes (Services) sa używane do eksponowania podów i umożliwienia im komunikacji. W naszej aplikacji wykorzystujemy usługi typu ClusterIP, które sa dostepne tylko wewnatrz klastra.

1.8.2 Wykorzystywane protokoły

Aplikacja korzysta z protokołów HTTP/HTTPS do komunikacji miedzy frontendem a backendem oraz miedzy backendem a innymi usługami (np. baza danych Redis). Proxy HTTP jest konfigurowane w serwerze Nginx, aby przekierowywać ruch do odpowiednich usług w klastrze.

1.8.3 Narzedzia do zarzadzania siecia

Do zarzadzania siecia w klastrze Kubernetes wykorzystujemy Ingress Controller, który jest konfigurowany za pomoca plików `ingress.yaml` oraz `ingress-class.yaml`. Ingress Controller umożliwia zewnetrzny dostep do usług aplikacji na podstawie reguł URL, co upraszcza zarzadzanie ruchem HTTP/HTTPS.

1.8.4 Izolacja sieciowa

Kubernetes zapewnia izolacje sieciowa miedzy podami za pomoca przestrzeni nazw (Namespaces) oraz polityk sieciowych (Network Policies). W naszej aplikacji, każdy komponent jest wdrażany w domyślnej przestrzeni nazw, a polityki sieciowe moga być używane do dodatkowego zabezpieczenia komunikacji miedzy podami.

Literatura

[1] Lei Gu and Huan Li, *Memory or time: Performance evaluation for iterative operation on hadoop and spark*, High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing (HPCC_EUC), 2013 IEEE 10th International Conference on, IEEE, 2013, pp. 721–727.

[2] Afshan K., *What is the difference between hadoop and spark?*, 2017.

[3] Amir K., *How do hadoop and spark stack up?*, 2018.

[4] R. Elmasri oraz S.B. Navathe, *Wprowadzenie do systemów baz danych*, Helion, 2019.