AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

Wydział Informatyki

INSTYTUT INFORMATYKI

Środowiska Udostępniania Usług

Grupa 4 - czwartek 9:45

Operator Framework - studium przypadku technologii

Operator Framework - a case study in technology

Dominika Bocheńczyk

Mateusz Łopaciński

Piotr Magiera

Michał Wójcik

Kraków, 2024

Spis treści

Rozdział 1

Wprowadzenie

Kubernetes to niekwestionowany lider w segmencie automatyzacji i orkiestracji aplikacji kontenerowych, pełniąc kluczową rolę w skalowalnym wdrażaniu oprogramowania na infrastrukturę sieciową. Doskonale wpisuje się w trend zastępowania architektur monolitycznych przez mikroserwisy, które znacząco zwiększają reaktywność i elastyczność systemów informatycznych. Istnieje jednakże pewien podzbiór aplikacji, dla którego pojawiają się wyzwania związane z automatyzacją ich obsługi, zwłaszcza w sytuacji awarii lub dynamicznego zwiększania obciążenia. Opisane aplikacje, to oprogramowanie typu *stateful*, takie jak bazy danych (Postgres, MySQL, Redis), middleware (RabbitMQ) czy systemy monitorowania (Prometheus), których stan jest krytyczny dla ciągłości działania i nie może zostać utracony w przypadku awarii.

Rysunek 1.1: Przykładowe aplikacje typu stateful

Zaproponowane przez twórców Kubernetes rozwiązania, takie jak Stateful Sets połączone z Persistent Volumes, umożliwiają utrzymanie danych na dysku i relacji master-slave między replikami baz danych, jednakże ich konfiguracja i zarządzanie mogą być złożone i czasochłonne, co utrudnia w pełni automatyczne zarządzanie cyklem życia tych aplikacji. Ponadto, standardowe narzędzia Kubernetes nie zawsze dostarczają wystarczających możliwości zarządzania stanem aplikacji, co może skutkować koniecznością utrzymywania systemów bazodanowych poza klastrem Kubernetes, co jest niezgodne z ideą Infrastructure as Code.

Rozdział 2

Podstawy teoretyczne i stos technologiczny

2.1. Podstawy teoretyczne

Operator Framework rozszerza możliwości Kubernetes, dostarczając narzędzia umożliwiające tworzenie operatorów – specjalistycznych programów, które zarządzają innymi aplikacjami wewnątrz klastra Kubernetes. Operatorzy są projektowani tak, aby w sposób ciągły monitorować stan aplikacji, automatycznie podejmując decyzje o koniecznych działaniach naprawczych, skalowaniu, aktualizacji lub konfiguracji w odpowiedzi na zmieniające się warunki operacyjne.

Istotą Operator Framework jest umożliwienie automatyzacji operacji, które tradycyjnie wymagałyby ręcznego przeprowadzenia przez zespoły operacyjne lub administratorów systemów. Przykładowo, operator bazy danych nie tylko zarządza replikacją danych, ale również może automatycznie zarządzać schematami bazy danych, przeprowadzać rotację certyfikatów, czy realizować procedury backupu i przywracania danych.

Jako że każda aplikacja typu stateful może posiadać specyficzny sposób zarządzania, potrzebuje ona swojego własnego Operatora. Z tego względu istnieje nawet publiczne repozytorium, z którego można pobrać konfiguracje opracowane pod konkretne oprogramowanie (znajduje się ono pod linkiem https://operatorhub.io/).

Wzorzec Operator pozwala na łączenie kontrolerów jednego lub więcej zasobów aby rozszerzyć zachowanie klastra bez konieczności zmiany implementacji. Operatorzy przyjmują rolę kontrolerów dla tzw. Custom Resource. Custom Resource rozszerzają / personalizują konkretne instalacje Kubernetesa, z tym zachowaniem że użytkownicy mogą z nich korzystać jak z wbudowanych już zasobów (np. Pods). [1]

2.2. Stos technologiczny

Głównym komponentem technologicznym naszego projketu jest Operator Framework, który zapewnia zestaw narzędzi, szablonów i wytycznych, ułatwiających programistom tworzenie operatorów, co przyspiesza proces tworzenia nowych aplikacji i usprawnia zarządzanie nimi w środowiskach Kubernetes.

W przypadku wyłącznie prezentacji działania Operatora na klastrze Kubernetesowym, możemy skorzystać z narzędzia jakim jest minikube. Minikube pozwala na szybkie stworzenie lokalnego klastra Kubernetesa w danym systemie operacyjnym. Dzięki temu, mając jedynie kontener Dockerowy lub środowisko maszyny wirtualnej, możemy lepiej skupić się na samej funkcjonalności Kubernetesa dla naszych potrzeb. [2]

Rozdział 3

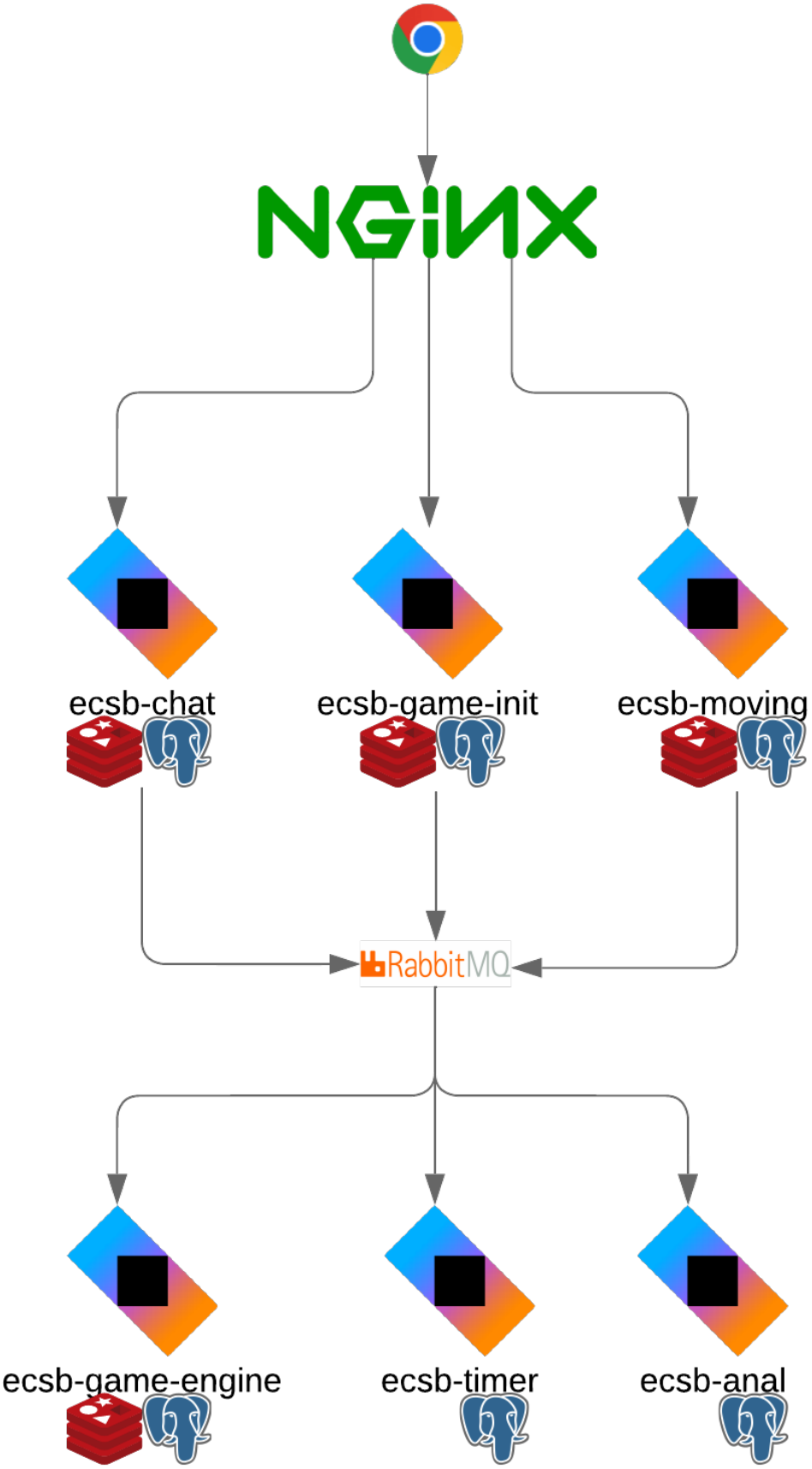
Opis studium przypadku

3.1. Przykład aplikacji typu stateful

Aplikacją, na której pokażemy działanie Operatora, będzie Elektroniczna Chłopska Szkoła Biznesu (eCSB), cyfrowa implementacja gry planszowej 'Chłopska Szkoła Biznesu' wydanej przez Małopolski Instytut Kultury. Aplikacja ta jest pracą inżynierską czworga studentów naszego Wydziału, obecnie kontynuowaną w ramach pracy magisterskiej. Gra polega na produkcji zasobów według przydzielonego zawodu, handlu towarami, zakładaniu spółek oraz odbywaniu wypraw w celu korzystniej wymiany produktów na pieniądze. W czasie kilkunastominutowej rozgrywki każdy z graczy stara się zgromadzić jak największy majątek.

Rysunek 3.1: Ekran powitalny ECSB

eCSB jest aplikacją webową opartą o mikroserwisy oraz komunikację z wykorzystaniem wiadomości. W minimalnym wariancie składa się z 6 bezstanowych modułów napisanych w języku Kotlin, podłączonych do bazy danych Postgres, bazy klucz-wartość Redis oraz brokera wiadomości RabbitMQ, a także aplikacji webowej, komunikującej się z modułami poprzez protokoły REST oraz WebSocket. Dostęp do wszystkich elementów architektury realizowany jest dzięki serwerowi HTTP nginx, który pełni rolę reverse-proxy (zapewniając przy tym certyfikaty SSL i ruch HTTPS) oraz API gateway (przekierowując żądania do odpowiednich mikroserwisów). Pełna architektura rozwiązania przedstawiona jest poniżej 4.1.



Rysunek 3.2: Architektura projektu eCSB

Warto dodać, że niektóre z modułów zostały zaprojektowane z myślą o skalowaniu systemu i rozkładaniu obciążenia. Są to moduły chat, moving oraz game-engine. Pozostałe 3 moduły odpowiadają za usługi stosunkowo rzadkie (tworzenie sesji gry) lub w pełni scentralizowane (zbieranie logów, odświeżanie czasu gry).

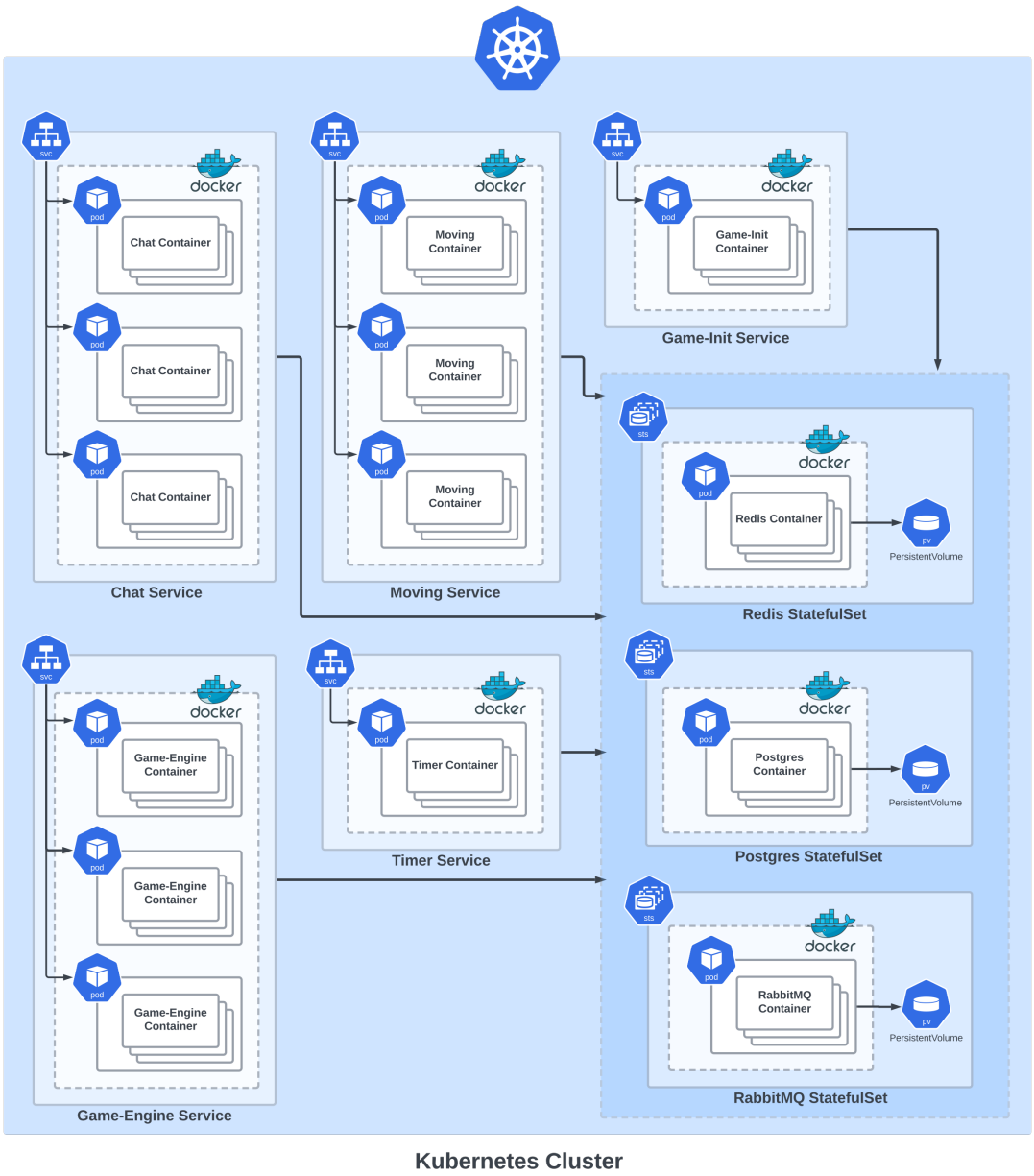
3.2. Przypadek użycia Operatora

Naszym scenariuszem, poprzez który zademonstrujemy działanie Operatora dla aplikacji eCSB, będą następujące operacje:

1. uruchomienie klastra Kubernetesowego z minimalnym wariantem eCSB (5 modułów - pomijamy moduł ecsb-anal, który służył jedynie do zbierania danych analitycznych)
2. potrojenie skalowalnych modułów (moving, chat, game-engine), a następnie powrót do wariantu minimalnego
3. przywrócenie centralnego modułu po awarii (timer lub game-init)

Rozdział 4

Architektura rozwiązania



Rysunek 4.1: Diagram architektury rozwiązania z wykorzystaniem Kubernetesa

Rozdział 5

Konfiguracja środowiska

Jako punkt wyjścia konfiguracji projektu został wyznaczony zmodyfikowany plik `docker-compose.yml`, za pomocą którego uruchamiane było middleware dla aplikacji backendowych oraz aplikacja webowa. Plik ten został rozszerzony o backendowe mikroserwisy oraz dodano wstępną konfigurację dla PostgreSQL oraz RabbitMQ:

```
FROM gradle:7-jdk11 AS build
COPY .. /home/gradle/ecsb-backend
WORKDIR /home/gradle/ecsb-backend
RUN gradle :ecsb-game-engine:buildFatJar --no-daemon

FROM openjdk:11
WORKDIR /home/gradle/ecsb-backend
RUN mkdir /app
COPY --from=build
   /home/gradle/ecsb-backend/ecsb-game-engine/build/libs/*.jar
   /app/ecsb-game-engine.jar
ENTRYPOINT ["java", "-jar", "/app/ecsb-game-engine.jar"]
```

Listing 5.1: Przykład pliku Dockerfile dla mikroserwisu

```
game-engine:
    container_name: game-engine
    build:
      context: ecsb-backend
      dockerfile: ./ecsb-game-engine/Dockerfile
    depends_on:
      - postgres
      - redis
      - rabbitmq
      - chat
    restart: on-failure
```

Listing 5.2: Przykładowa definicja kontenera mikroserwisu w pliku docker-compose.yml

Mając z grubsza zdefiniowane kontenery składające się na architekturę, następnym krokiem było opakowanie ich w Kubernetesowe serwisy oraz konfiguracja zmiennych środowiskowych.

5.1. Konfiguracja w Kubernetes

W celu przeniesienia konfiguracji do Kubernetes, należy stworzyć odpowiednie pliki YAML definiujące zasoby takie jak ConfigMap, Pod, Deployment, Service oraz PersistentVolumeClaim. Przykładowe pliki konfiguracyjne:

5.1.1. ConfigMap dla RabbitMQ

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: rabbitmq
data:
  enabled-plugins: "[rabbitmq_management,rabbitmq_sharding]."
  rabbitmq-conf: |
    vm_memory_high_watermark.relative = 0.4
    vm_memory_calculation_strategy = rss
```

Listing 5.3: ConfigMap dla RabbitMQ

5.1.2. Pod dla mikroserwisu

```
apiVersion: v1
kind: Pod
metadata:
  name: chat
  labels:
    app.kubernetes.io/name: chat
spec:
  containers:
    - image: michwoj01/ecsb:chat
      name: chat
      ports:
        - containerPort: 2138
      resources: {}
  restartPolicy: OnFailure
status: {}
```

Listing 5.4: Pod dla mikroserwisu

5.1.3. Deployment dla RabbitMQ

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: rabbitmq
  labels:
    app.kubernetes.io/name: rabbitmq
spec:
  replicas: 1
  selector:
    matchLabels:
```

```
      app.kubernetes.io/name: rabbitmq
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app.kubernetes.io/name: rabbitmq
    spec:
      containers:
        - image: rabbitmq:3.12-management
          name: rabbitmq
          ports:
            - containerPort: 15672
            - containerPort: 5672
          resources: {}
          volumeMounts:
            - mountPath: /etc/rabbitmq/
              name: rabbitmq
      restartPolicy: Always
      volumes:
        - name: rabbitmq
          configMap:
            name: rabbitmq
            items:
            - key: rabbitmq-conf
              path: rabbitmq.conf
            - key: enabled-plugins
              path: enabled_plugins
status: {}
```

Listing 5.5: Deployment dla RabbitMQ

5.1.4. Service dla PostgreSQL

```
apiVersion: v1
kind: Service
metadata:
  name: postgres
spec:
  ports:
    - name: "5432"
      port: 5432
      targetPort: 5432
  selector:
    app.kubernetes.io/name: postgres
status:
  loadBalancer: {}
```

Listing 5.6: Service dla PostgreSQL

5.1.5. PersistentVolumeClaim dla PostgreSQL

```
apiVersion: v1
kind: PersistentVolumeClaim
```

```
metadata:
  name: postgres-claim0
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 100Mi
status: {}
```

Listing 5.7: PersistentVolumeClaim dla PostgreSQL

Konfiguracja w Kubernetes pozwala na zarządzanie mikroserwisami w bardziej skalowalny i elastyczny sposób, umożliwiając dynamiczne skalowanie, zarządzanie stanem aplikacji oraz łatwiejszą integrację z innymi komponentami systemu. Wszystkie pliki konfiguracyjne YAML powinny być umieszczone w odpowiednim katalogu (np. `kubernetes`), aby można było łatwo zastosować je w klastrze Kubernetes.

Rozdział 6

Sposób instalacji

6.1. Instalacja niezbędnych narzędzi

1. **Instalacja Docker**:
   (a) Należy przejść na stronę Docker Desktop i pobrać odpowiednią wersję dla systemu operacyjnego (Windows, macOS, Linux).
   (b) Następnie należy postępować zgodnie z opisanymi krokami instalacji dla używanego systemu operacyjnego.
2. **Instalacja Minikube**:
   (a) Należy przejść na stronę Minikube i pobrać odpowiednią wersję dla systemu operacyjnego, a następnie postępować zgodnie z krokami instalacji opisanymi w na załączonej stronie internetowej.
   (b) Uruchomienie Minikube:

```
minikube start
```

6.2. Uruchomienie aplikacji na Kubernetes

1. **Zastosowanie wszystkich plików konfiguracyjnych Kubernetes**:

```
minikube kubectl -- apply -f kubernetes -R
```

   Jeżeli napotkano błąd `error looking up service account default/default: serviceaccount "default"not found`, ponowne uruchomienie powyższej komendy.
2. **Sprawdzenie statusu**:

```
minikube kubectl -- get pods
```

6.3. Debugowanie

W przypadku problemów, sprawdzenie logów podów:

```
minikube kubectl -- logs <pod-name>
```

Rozdział 7

Odtworzenie rozwiązania

W tym rozdziale opisany jest proces odtworzenia całego rozwiązania krok po kroku, z wykorzystaniem podejścia Infrastructure as Code (IaC). Podejście to znacząco ułatwia proces wdrażania infrastruktury, poprzez jego automatyzację i ogranicza się do wykonania kilku komend, które, na podstawie zdefiniowanych plików konfiguracyjnych, wykonają niezbędną pracę za nas.

7.1. Infrastructure as Code (IaC) - Podejście

Infrastructure as Code (IaC) to praktyka zarządzania infrastrukturą IT przy użyciu kodu, co pozwala na automatyzację procesów wdrożeniowych oraz łatwe zarządzanie środowiskiem. W naszym projekcie wykorzystujemy Kubernetes do zarządzania kontenerami oraz Docker do budowy obrazów kontenerów.

7.2. Kroki odtworzenia rozwiązania

1. **Instalacja narzędzi**: Na wstępie należy zainstalować niezbędne narzędzia, takie jak Docker, Minikube oraz kubectl zgodnie z instrukcjami podanymi w rozdziale dotyczącym instalacji.

2. **Klonowanie repozytorium projektu**: Aby rozpocząć pracę, należy sklonować repozytorium projektu zawierające wszystkie niezbędne pliki konfiguracyjne i kody źródłowe:

```
git clone
   https://github.com/michwoj01/EOSI-Operator-Framework.git
cd EOSI-Operator-Framework
```

3. **Budowanie obrazów z wykorzystaniem Dockera**: Następnie, należy zbudować obrazy Docker dla wszystkich mikroserwisów zgodnie z konfiguracją zawartą w docker-compose.yml.

4. **Zastosowanie konfiguracji Kubernetesa**: Wszystkie pliki konfiguracyjne Kubernetes znajdują się w katalogu `kubernetes`. Aby zastosować te konfiguracje, używamy komendy:

```
minikube kubectl -- apply -f kubernetes -R
```

5. **Konfiguracja RabbitMQ i PostgreSQL**: RabbitMQ jest konfigurowany przy użyciu ConfigMap, a PostgreSQL przy użyciu PersistentVolumeClaim, aby zapewnić trwałe przechowywanie danych. Szczegóły konfiguracji znajdują się w odpowiednich plikach YAML.

6. **Tworzenie podów i usług**: Konfiguracja podów i usług dla wszystkich komponentów aplikacji (mikroserwisów, baz danych, kolejek) jest zdefiniowana w plikach YAML. Każdy plik definiuje specyfikację dla odpowiedniego komponentu, w tym zmienne środowiskowe, porty i zależności.

7. **Sprawdzenie stanu klastra**: Po zastosowaniu konfiguracji, należy sprawdzić status uruchomionych podów, aby upewnić się, że wszystkie komponenty zostały poprawnie wdrożone:

```
minikube kubectl -- get pods
```

8. **Debugowanie**: W przypadku problemów, logi podów mogą dostarczyć niezbędnych informacji do diagnozowania i rozwiązywania problemów:

```
minikube kubectl -- logs <pod-name>
```

9. **Dostęp do aplikacji**: Po poprawnym uruchomieniu wszystkich usług, aplikacja powinna być dostępna. Minikube dostarcza polecenie, które pozwala na dostęp do usług zewnętrznych poprzez otwarcie odpowiednich portów:

```
minikube service <service-name>
```

Rozdział 8

Wdrożenie wersji demonstracyjnej

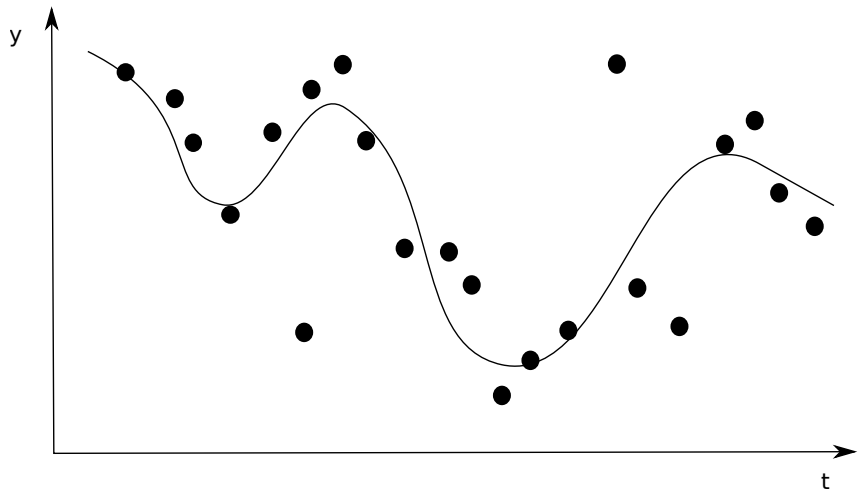
Rozdział 9

Podsumowanie

9.1. Rysunki, tabele

9.1.1. Rysunki

Przykładowy odnośnik do rysunku 9.1.



Rysunek 9.1: Przykładowy rysunek (źródło: [5])

W przypadku rysunków można odwoływać się zarówno do poszczególnych części składowych — rysunek 9.2(a) i rysunek 9.2(b)) — jak i do całego rysunku 9.2.

9.1.2. Tabele

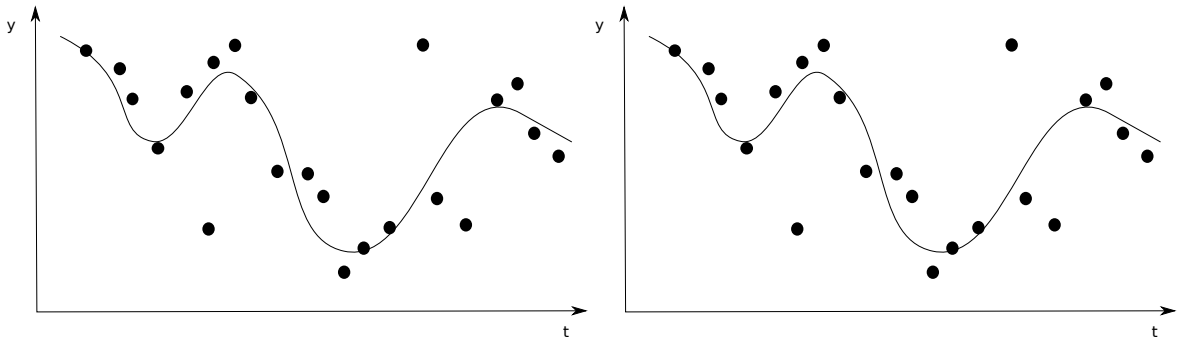
Przykładowa tabela 9.1.

9.2. Wzory matematyczne

Przykład wzoru z odnośnikiem do literatury [5]:

$$\Omega = \sum_{i=1}^{n} \gamma_i \tag{9.1}$$

Przykładowy odnośnik do wzoru (9.1).



(a) Tytuł 1 (b) Tytuł 2

Rysunek 9.2: Kolejne przykładowe rysunki (źródło: [5])

Tabela 9.1: Przykładowa tabela

	Best		**Average**		**Worst**	
No.	**AB**	**CD**	**FE**	**GH**	**IJ**	**KL**
1.	10	89	58	244	6	70
2.	15	87	57	147	4	82
3.	23	45	55	151	2	38
4.	34	90	55	246	1	82
5.	56	75	54	255	0	73

Przykładowy wzór w tekście $\lambda = \sum_{i=1}^{n} \delta_i$, bez numeracji.

9.3. Cytowania

Przykład cytowania literatury [9]. Kolejny przykład cytowania kilku pozycji bibliograficznych [3, 10, 7, 8, 4].

9.4. Listy

Lista z elementami:

- pierwszym,
- drugim,
- trzecim.

Lista numerowana z dłuższymi opisami:

1. Pierwszy element listy.
2. Drugi element listy.
3. Trzeci element listy.

9.5. Algorytmy

Algorytm 1 przedstawia przykładowy algorytm zaprezentowany w [6].

Algorytm 1: Przykładowy algorytm (źródło: [6]).

```
   input  : A bitmap Im of size w × l
   output  A partition of the bitmap
   :
 1 special treatment of the first line;
 2 for i ← 2 to l do
 3     special treatment of the first element of line i;
 4     for j ← 2 to w do
 5         left ← FindCompress(Im[i, j − 1]);
 6         up ← FindCompress(Im[i − 1, ]);
 7         this ← FindCompress(Im[i, j]);
 8         if left compatible with this then // O(left,this)==1
 9             if left < this then Union(left,this);
10             else Union(this,left);
11         end
12         if up compatible with this then                    // O(up,this)==1
13             if up < this then Union(up,this);
               // this is put under up to keep tree as flat as possible
14             else Union(this,up);
               // this linked to up
15         end
16     end
17     foreach element e of the line i do FindCompress(p);
18 end
```

9.6. Fragmenty kodu źródłowego

Listing 9.1 przedstawia przykładowy fragment kodu źródłowego.

```
# The maximum of two numbers

def maximum(x, y):

    if x >= y:
        return x
    else:
        return y

x = 2
y = 6
print(maximum(x, y),"is the largest of the numbers ", x, " and ", y)
```

Listing 9.1: Przykładowy fragment kodu (źródło: [5])

Bibliografia

[1] 2024. URL: https://kubernetes.io/docs/concepts/extend-kubernetes/operator/.

[2] 2024. URL: https://minikube.sigs.k8s.io/docs/start/.

[3] F. Allen i R. Karjalainen. „Using Genetic Algorithms to Find Technical Trading Rules". W: *Journal of Financial Economics* 51.2 (1999), s. 245–271.

[4] G. Chmaj i H. Selvaraj. „Distributed Processing Applications for UAV/Drones: A Survey". W: *Progress in Systems Engineering. Advances in Intelligent Systems and Computing*. Red. C. G. Selvaraj H. Zydek D. T. 366. Springer, Cham, 2015, s. 449–454. DOI: 10.1007/978-3-319-08422-0_66.

[5] A. Exemplary. *Exemplary title of the book*. Address of the publisher: Publisher, 2021.

[6] C. Fiorio. *algorithm2e.sty – package for algorithms*. 2017. URL: http://mirrors.ctan.org/macros/latex/contrib/algorithm2e/doc/algorithm2e.pdf.

[7] V. Pictet O. *Genetic Algorithms with Collective Sharing for Robust Optimization in Financial Applications*. Technical report. Olsen & Associates, 1995.

[8] F. Wilhelmstötter. *Jenetics*. 2021. URL: https://jenetics.io/ (term. wiz. 28. 01. 2021).

[9] G. Wilson i W. Banzhaf. „Prediction of Interday Stock Prices using Developmental and Linear Genetic Programming". W: *Applications of Evolutionary Computing. EvoWorkshops 2009: EvoCOMNET, EvoENVIRONMENT, EvoFIN, EvoGAMES, EvoHOT, EvoIASP, EvoINTERACTION, EvoMUSART, EvoNUM, EvoSTOC, EvoTRANSLOG, Tübingen, Germany, April 15-17, 2009, Proceedings*. Red. M. Giacobini i in. T. 5484. LNCS. Berlin, Heidelberg: Springer-Verlag, 2009, s. 172–181.

[10] E. Zitzler. „Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications". PhD thesis. ETH Zurich, 1999.

Spis rysunków

Spis tabel

Spis algorytmów

Spis listingów