O'REILLY

Wydanie II

Kubernetes

Tworzenie niezawodnych systemów rozproszonych

Brendan Burns
Joe Beda
Kelsey Hightower

Tytuł oryginału: Kubernetes: Up and Running: Dive into the Future of Infrastructure, 2nd Edition

Tłumaczenie: Łukasz Piwko z wykorzystaniem fragmentów książki „Kubernetes. Tworzenie niezawodnych systemów rozproszonych" w przekładzie Lecha Lachowskiego

ISBN: 978-83-283-6745-6

© 2020 Helion SA
Authorized Polish translation of the English edition of Kubernetes: Up and Running, 2nd Edition ISBN 9781492046530 © 2019 Brendan Burns, Joe Beda, and Kelsey Hightower

This translation is published and sold by permission of O'Reilly Media, Inc., which owns or controls all rights to publish and sell the same.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from the Publisher.

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej publikacji w jakiejkolwiek postaci jest zabronione. Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

Wszystkie znaki występujące w tekście są zastrzeżonymi znakami firmowymi bądź towarowymi ich właścicieli.

Autor oraz Helion SA dołożyli wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie biorą jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Autor oraz Helion SA nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *http://helion.pl* (księgarnia internetowa, katalog książek)

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
ftp://ftp.helion.pl/przyklady/kuber2.zip

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/kuber2
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

• Kup książkę
• Poleć książkę
• Oceń książkę

• Księgarnia internetowa
• Lubię to! » Nasza społeczność

Spis treści

Przedmowa 13

1. Wprowadzenie 19
Prędkość 20
Wartość niemutowalności 20
Deklaratywna konfiguracja 22
Systemy samonaprawiające się 22
Skalowanie usługi i zespołów programistycznych 23
Rozłączność 23
Łatwe skalowanie aplikacji i klastrów 24
Skalowanie zespołów programistycznych za pomocą mikrousług 24
Separacja zagadnień dla zapewnienia spójności i skalowania 25
Zapewnianie abstrakcji infrastruktury 27
Wydajność 27
Podsumowanie 28

2. Tworzenie i uruchamianie kontenerów 29
Obrazy kontenerów 30
Format obrazu Dockera 30
Budowanie obrazów aplikacji za pomocą Dockera 32
Pliki Dockerfile 32
Optymalizacja rozmiarów obrazu 34
Bezpieczeństwo obrazu 35
Wieloetapowe budowanie obrazów 35
Przechowywanie obrazów w zdalnym rejestrze 36
Środowisko wykonawcze kontenera Dockera 37
Uruchamianie kontenerów za pomocą Dockera 38
Odkrywanie aplikacji kuard 38
Ograniczanie wykorzystania zasobów 38
Czyszczenie 39
Podsumowanie 40

Kup ksi k   Pole ksi k

3. Wdrażanie klastra Kubernetes 41
Instalowanie Kubernetes w usłudze dostawcy publicznej chmury 41
Google Kubernetes Engine 42
Instalowanie Kubernetes w Azure Kubernetes Service 42
Instalowanie Kubernetes w Amazon Web Services 43
Lokalna instalacja Kubernetes za pomocą minikube 43
Uruchamianie Kubernetes w Dockerze 44
Uruchamianie Kubernetes na Raspberry Pi 44
Klient Kubernetes 44
Sprawdzanie statusu klastra 45
Wyświetlanie węzłów roboczych klastra Kubernetes 45
Komponenty klastra 47
Serwer proxy Kubernetes 47
Serwer DNS Kubernetes 48
Interfejs użytkownika Kubernetes 48
Podsumowanie 49

4. Typowe polecenia kubectl 51
Przestrzenie nazw 51
Konteksty 51
Przeglądanie obiektów interfejsu API Kubernetes 52
Tworzenie, aktualizacja i niszczenie obiektów Kubernetes 52
Dodawanie etykiet i adnotacji do obiektów 53
Polecenia debugowania 54
Uzupełnianie poleceń 55
Inne sposoby pracy z klastrami 56
Podsumowanie 56

5. Kapsuły 57
Kapsuły w Kubernetes 58
Myślenie w kategoriach kapsuł 58
Manifest kapsuły 59
Tworzenie kapsuły 59
Tworzenie manifestu kapsuły 60
Uruchamianie kapsuł 61
Wyświetlanie listy kapsuł 61
Szczegółowe informacje o kapsule 61
Usuwanie kapsuły 62
Uzyskiwanie dostępu do kapsuły 63
Korzystanie z przekierowania portów 63
Uzyskiwanie większej ilości informacji za pomocą dzienników 63

Kup ksi k   Pole ksi k

Uruchamianie poleceń w kontenerze przy użyciu exec 64
Kopiowanie plików do i z kontenerów 64
Kontrole działania 65
Sonda żywotności 65
Sonda gotowości 66
Rodzaje kontroli działania 66
Zarządzanie zasobami 67
Żądania zasobów: minimalne wymagane zasoby 67
Ograniczanie wykorzystania zasobów za pomocą limitów 69
Utrwalanie danych za pomocą woluminów 69
Używanie woluminów z kapsułami 70
Różne sposoby używania woluminów z kapsułami 70
Utrwalanie danych przy użyciu dysków zdalnych 71
Wszystko razem 72
Podsumowanie 73

6. Etykiety i adnotacje 75
Etykiety 76
Stosowanie etykiet 76
Modyfikowanie etykiet 77
Selektory etykiet 78
Selektory etykiet w obiektach API 80
Etykiety w architekturze Kubernetes 80
Adnotacje 81
Definiowanie adnotacji 82
Czyszczenie 82
Podsumowanie 82

7. Wykrywanie usług 83
Co to jest wykrywanie usług? 83
Obiekt Service 84
DNS usługi 85
Kontrole gotowości 85
Udostępnianie usługi poza klastrem 87
Integracja z chmurą 88
Szczegóły dla zaawansowanych 89
Punkty końcowe 89
Ręczne wykrywanie usług 90
kube-proxy i adresy IP klastra 91
Zmienne środowiskowe adresu IP klastra 91
Łączenie z innymi środowiskami 92
Czyszczenie 93
Podsumowanie 93

Kup ksi k Pole ksi k

8. Równoważenie obciążenia HTTP przy użyciu Ingress 95
Specyfikacja Ingress i kontrolery Ingress 96
Instalacja Contour 96
Konfiguracja DNS 97
Konfiguracja pliku lokalnych hostów 98
Praca z Ingress 98
Najprostszy sposób użycia 99
Używanie nazw hosta 99
Ścieżki 101
Czyszczenie 101
Techniki zaawansowane i pułapki 101
Uruchamianie kilku kontrolerów Ingress 102
Wiele obiektów Ingress 102
Ingress i przestrzenie nazw 102
Przepisywanie ścieżek 103
Serwowanie przez TLS 103
Inne implementacje Ingress 104
Przyszłość Ingress 105
Podsumowanie 105

9. Obiekt ReplicaSet 107
Pętle uzgadniania 108
Relacje między kapsułami i obiektami ReplicaSet 108
Adaptowanie istniejących kontenerów 109
Poddawanie kontenerów kwarantannie 109
Projektowanie z wykorzystaniem ReplicaSet 109
Specyfikacja ReplicaSet 109
Szablony kapsuł 110
Etykiety 110
Tworzenie obiektu ReplicaSet 111
Inspekcja obiektu ReplicaSet 111
Znajdowanie ReplicaSet z poziomu kapsuły 112
Znajdowanie zestawu kapsuł dla ReplicaSet 112
Skalowanie kontrolerów ReplicaSet 112
Skalowanie imperatywne za pomocą polecenia kubectl scale 112
Skalowanie deklaratywne za pomocą kubectl apply 113
Automatyczne skalowanie kontrolera ReplicaSet 113
Usuwanie obiektów ReplicaSet 115
Podsumowanie 115

Kup ksi k Pole ksi k

10. Obiekt Deployment 117
Twoje pierwsze wdrożenie 118
Wewnętrzne mechanizmy działania obiektu Deployment 118
Tworzenie obiektów Deployment 119
Zarządzanie obiektami Deployment 121
Aktualizowanie obiektów Deployment 122
Skalowanie obiektu Deployment 122
Aktualizowanie obrazu kontenera 122
Historia wersji 123
Strategie wdrażania 126
Strategia Recreate 126
Strategia RollingUpdate 126
Spowalnianie wdrażania w celu zapewnienia poprawnego działania usługi 129
Usuwanie wdrożenia 130
Monitorowanie wdrożenia 131
Podsumowanie 131

11. Obiekt DaemonSet 133
Planista DaemonSet 134
Tworzenie obiektów DaemonSet 134
Ograniczanie użycia kontrolerów DaemonSet do określonych węzłów 136
Dodawanie etykiet do węzłów 136
Selektory węzłów 137
Aktualizowanie obiektu DaemonSet 138
Ciągła aktualizacja obiektu DaemonSet 138
Usuwanie obiektu DaemonSet 139
Podsumowanie 139

12. Obiekt Job 141
Obiekt Job 141
Wzorce obiektu Job 141
Zadania jednorazowe 142
Równoległość 146
Kolejki robocze 147
Obiekt CronJob 151
Podsumowanie 151

13. Obiekty ConfigMap i tajne dane 153
Obiekty ConfigMap 153
Tworzenie obiektów ConfigMap 153
Używanie obiektów ConfigMap 154

Kup ksi k Pole ksi k

Tajne dane 156
Tworzenie tajnych danych 157
Korzystanie z tajnych danych 158
Prywatne rejestry Dockera 160
Ograniczenia dotyczące nazewnictwa 160
Zarządzanie obiektami ConfigMap i tajnymi danymi 161
Wyświetlanie obiektów 161
Tworzenie obiektów 162
Aktualizowanie obiektów 162
Podsumowanie 164

14. Model kontroli dostępu oparty na rolach w Kubernetes ... 165
Kontrola dostępu oparta na rolach 166
Tożsamość w Kubernetes 166
Role i powiązania ról 167
Role i powiązania ról w Kubernetes 167
Techniki zarządzania funkcją RBAC 169
Testowanie autoryzacji za pomocą narzędzia can-i 169
Zarządzanie funkcją RBAC w kontroli źródła 169
Tematy zaawansowane 170
Agregowanie ról klastrowych 170
Wykorzystywanie grup do wiązań 171
Podsumowanie 172

15. Integracja rozwiązań do przechowywania danych i Kubernetes ... 173
Importowanie usług zewnętrznych 174
Usługi bez selektorów 175
Ograniczenia usług zewnętrznych: sprawdzanie poprawności działania 176
Uruchamianie niezawodnych singletonów 176
Uruchamianie singletona MySQL 177
Dynamiczne przydzielanie woluminów 180
Natywne magazyny danych Kubernetes z wykorzystaniem obiektów StatefulSet 181
Właściwości obiektów StatefulSet 181
Ręcznie zreplikowany klaster MongoDB z wykorzystaniem obiektów StatefulSet 182
Automatyzacja tworzenia klastra MongoDB 184
Trwałe woluminy i obiekty StatefulSet 186
Ostatnia rzecz: sondy gotowości 187
Podsumowanie 187

Kup ksi k Pole ksi k

16. Rozszerzanie Kubernetes 189
Co znaczy rozszerzanie Kubernetes 189
Punkty rozszerzalności 190
Wzorce tworzenia zasobów 197
Tylko dane 197
Kompilatory 197
Operatory 198
Jak zacząć 198
Podsumowanie 198

17. Wdrażanie rzeczywistych aplikacji 199
Jupyter 199
Parse 200
Wymagania wstępne 201
Budowanie serwera parse-server 201
Wdrażanie serwera parse-server 201
Testowanie Parse 202
Ghost 202
Konfigurowanie serwera Ghost 203
Redis 205
Konfigurowanie instalacji Redis 206
Tworzenie usługi Redis 207
Wdrażanie klastra Redis 208
Zabawa z klastrem Redis 209
Podsumowanie 210

18. Organizacja aplikacji 211
Podstawowe zasady 211
Systemy plików jako źródło prawdy 211
Rola recenzji kodu 212
Bramy i flagi funkcji 212
Zarządzanie aplikacją w systemie kontroli źródła 213
Układ systemu plików 213
Wersje okresowe 214
Konstruowanie aplikacji
w sposób umożliwiający jej rozwój, testowanie i wdrażanie 216
Cele 216
Progresja wydania 216
Parametryzacja aplikacji za pomocą szablonów 217
Parametryzacja przy użyciu narzędzia Helm i szablonów 218
Parametryzacja systemu plików 218

Kup ksi k Pole ksi k

Wdrażanie aplikacji na całym świecie 219
Architektura umożliwiająca wdrażanie aplikacji na całym świecie 219
Implementacja wdrożenia światowego 220
Pulpity i monitorowanie wdrożeń światowych 221
Podsumowanie 222

A Budowanie klastra Raspberry Pi Kubernetes 223
Lista części 223
Flashowanie obrazów 224
Pierwsze uruchomienie: węzeł główny 224
Konfigurowanie sieci 225
Instalowanie Kubernetes 226
Konfigurowanie klastra 227
Podsumowanie 228

Kup ksi k Pole ksi k

ROZDZIAŁ 16.

Rozszerzanie Kubernetes

Od samego początku było jasne, że Kubernetes nie zostanie na poziomie podstawowego zestawu interfejsów API. Po wprowadzeniu aplikacji do klastra użytkownik ma do wyboru mnóstwo dodatkowych narzędzi, które może wdrożyć w Kubernetes jako obiekty API. Trzeba było tylko zastanowić się, jak zapanować nad tą obfitością obiektów i zastosowań, nie dopuszczając do rozrostu API do gigantycznych rozmiarów.

Aby rozwiązać problem związany z puchnięciem API przy dodawaniu rozszerzeń, poświęcono wiele pracy na to, by interfejs API Kubernetes uczynić rozszerzalnym. Umożliwiło to operatorom klastrów rozszerzanie funkcjonalności za pomocą komponentów spełniających ich wymagania. Wprowadzony model pozwala na samodzielne rozszerzanie funkcjonalności klastrów, wykorzystywanie dodatków udostępnianych przez członków społeczności oraz tworzenie rozszerzeń, które można sprzedawać w postaci wtyczek. Ponadto możliwość wprowadzania rozszerzeń przyczyniła się do powstania nowych wzorców zarządzania systemami, takich jak na przykład wzorzec operatora.

Znajomość mechanizmów rozszerzania API serwera Kubernetes oraz technik tworzenia i dostarczania rozszerzeń jest niezbędna do optymalnego wykorzystania możliwości tego systemu i jego środowiska niezależnie od tego, czy tworzy się własne rozszerzenia, czy korzysta się tylko z gotowych rozwiązań. W miarę jak dzięki tym funkcjom powstają coraz bardziej zaawansowane narzędzia i platformy na bazie Kubernetes, praktyczna znajomość sposobu ich działania staje się niezbędna dla każdego, kto chce budować aplikacje w nowoczesnym klastrze Kubernetes.

Co znaczy rozszerzanie Kubernetes

Generalnie rozszerzenia serwera API Kubernetes dodają nowe funkcje do klastra lub ograniczają i modyfikują sposoby interakcji użytkowników z klastrami. Administratorzy klastrów mają do dyspozycji szeroki wybór wtyczek, które rozszerzają ich klastry o nowe funkcje i usługi. Należy podkreślić, że klaster mogą rozszerzać tylko użytkownicy o bardzo wysokich uprawnieniach. Nie jest to czynność dostępna dla przypadkowych użytkowników lub programów, ponieważ wymaga uprawnień administratora. Nawet administratorzy klastra powinni być bardzo ostrożni w trakcie instalowania narzędzi zewnętrznych. Niektóre rozszerzenia, takie jak kontrolery wstępu, pozwalają przeglądać wszystkie obiekty tworzone w klastrze, przez co mogą zostać wykorzystane do kradzieży tajnych kodów i uruchamiania złośliwego kodu. Ponadto klaster z rozszerzeniami różni się od podstawowego klastra Kubernetes. Jeśli liczba klastrów jest większa, warto opracować narzędzia umożliwiające zapewnienie spójności sposobu pracy z nimi. Dotyczy to także instalowanych rozszerzeń.

Kup ksi k

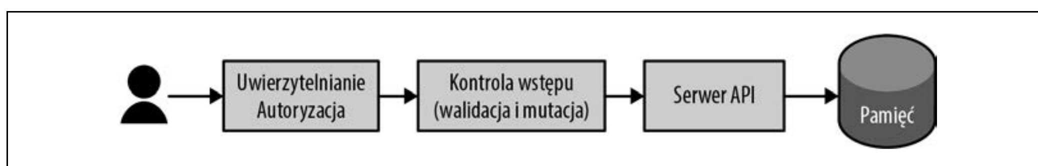
Pole ksi k

Punkty rozszerzalności

Kubernetes można rozszerzać na wiele sposobów, na przykład przy użyciu własnych definicji zasobów (`CustomResourceDefinition`) lub wtyczek Container Network Interface (CNI). W tym rozdziale skupiamy się na dodawaniu nowych typów zasobów do serwera API lub kontrolerów wstępu do żądań API. Nie opisujemy rozszerzeń CNI/CSI/CRI (ang. *Container Network Interface/Container Storage Interface/Container Runtime Interface*), ponieważ z nich częściej korzystają twórcy klastrów Kubernetes, a książka ta jest przeznaczona dla użytkowników końcowych tego systemu.

Oprócz kontrolerów wstępu i rozszerzeń API istnieją też narzędzia „rozszerzania" klastra w ogóle niewymagające modyfikacji serwera API. Zaliczają się do nich na przykład zasoby `DaemonSet` instalujące automatyczne funkcje zapisu danych w dzienniku i monitoringu, narzędzia skanujące usługi w poszukiwaniu luk bezpieczeństwa umożliwiających przeprowadzanie ataków XSS (ang. *cross-site scripting*) i wiele więcej. Zanim jednak zaczniesz rozszerzać klaster, dobrze zapoznaj się z możliwościami podstawowych API Kubernetes.

W zrozumieniu roli kontrolerów wstępu i zasobów `CustomResourceDefinition` pomocna jest wiedza na temat przepływu żądań przez serwer API Kubernetes, który jest pokazany na rysunku 16.1.



Rysunek 16.1. Przepływ żądań przez serwer API

Kontrolery wstępu, które są wywoływane przed zapisaniem obiektu API w pamięci, mogą odrzucać lub modyfikować żądania API. W serwer API Kubernetes jest wbudowanych kilka kontrolerów wstępu. Na przykład kontroler ograniczeń zakresu określa domyślne limity dla kapsuł, które ich nie mają. Wiele innych systemów wykorzystuje własne kontrolery wstępu do automatycznego wstrzykiwania pobocznych kontenerów do wszystkich kapsuł utworzonych w systemie w celu włączenia „magicznych automatyzacji".

Inny rodzaj rozszerzenia, który także może być stosowany w połączeniu z kontrolerami wstępu, to zasoby własne użytkownika. Przy ich użyciu dodaje się całe nowe obiekty do API Kubernetes. Obiekty te można wprowadzać do przestrzeni nazw, podlegają funkcji RBAC, a dostęp do nich jest możliwy zarówno poprzez standardowe narzędzia, jak `kubectl`, jak i przez API Kubernetes.

Dalej bardziej szczegółowo opisujemy te typy rozszerzeń klastrów oraz przedstawiamy praktyczne przykłady ich zastosowania.

Przygotowanie własnego zasobu należy zacząć od utworzenia obiektu `CustomResourceDefinition`, który jest tak zwanym metazasobem, czyli zasobem, który definiuje inny zasób.

W ramach przykładu zdefiniujemy nowy zasób reprezentujący testy obciążeniowe klastra. Gdy zostanie utworzony zasób `LoadTest`, w klastrze Kubernetes nastąpi uruchomienie testu obciążenia i skierowanie ruchu do usługi.

Kup ksi k Pole ksi k

Tworzenie nowego zasobu zaczynamy od jego zdefiniowania za pomocą obiektu `CustomResource` ↪`Definition`. Poniżej znajduje się przykładowa definicja:

```
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
  name: loadtests.beta.kuar.com
spec:
  group: beta.kuar.com
  versions:
    - name: v1
      served: true
      storage: true
  scope: Namespaced
  names:
    plural: loadtests
    singular: loadtest
    kind: LoadTest
    shortNames:
    - lt
```

Jak widać, jest to typowy obiekt Kubernetes zawierający podobiekt `metadata`, w którym znajduje się nazwa zasobu. Jednak w zasobach użytkownika nazwa jest wyjątkowa, ponieważ musi mieć następujący format: *`<nazwawliczbiemnogiej>.<grupa-api>`*. Zgodność nazw wszystkich obiektów `Custom` ↪`ResourceDefinition` z tym wzorem zapewnia niepowtarzalność definicji zasobów, ponieważ żadne dwa obiekty w klastrze nie mogą mieć takiej samej nazwy. To daje nam gwarancję, że nie ma dwóch definicji `CustomResourceDefinitions` opisujących ten sam zasób.

Oprócz metadanych definicja `CustomResourceDefinition` zawiera też podobiekt `spec`, w którym znajduje się właśnie definicja samego zasobu. Należące do niego pole `apigroup` określa grupę API zasobu. Jak napisaliśmy wcześniej, musi ona odpowiadać sufiksowi nazwy obiektu `CustomResource` ↪`Definition`. Ponadto w tej części znajduje się lista wersji zasobu. Zawiera ona nazwę wersji (na przykład `v1`, `v2` itd.) oraz pola określające, czy dana wersja jest obsługiwana przez serwer API i która wersja jest używana do zapisywania danych w pamięci serwera API. Pole `storage` może mieć wartość `true` tylko dla jednej wersji zasobu. Pole `scope` określa, czy dany zasób należy do przestrzeni nazw, czy nie (domyślnie tak), a w polu `names` można podać nazwy w liczbie pojedynczej, mnogiej i nazwę rodzajową. Można też zdefiniować „krótkie nazwy", którymi łatwiej jest posługiwać się w `kubectl` i w innych narzędziach.

Na podstawie tej definicji można utworzyć zasób w serwerze API Kubernetes. Najpierw jednak wyświetlimy listę naszych zasobów `loadtests`, aby pokazać prawdziwą naturę dynamicznych typów zasobów:

```
$ kubectl get loadtests
```

Dowiesz się, że w tej chwili taki zasób jest niezdefiniowany.

Teraz go utworzymy przy użyciu pliku *loadtest-resource.yaml*:

```
$ kubectl create -f loadtest-resource.yaml
```

Ponownie pobieramy listę zasobów `loadtests`:

```
$ kubectl get loadtests
```


Kup ksi k Pole ksi k

Tym razem zostaje znaleziona definicja zasobu typu `LoadTest`, choć nadal nie ma żadnych jego egzemplarzy.

Zmienimy to, tworząc nowy zasób `LoadTest`.

Własny zasób (w tym przypadku `LoadTest`), podobnie jak wszystkie wbudowane obiekty API Kubernetes, można zdefiniować w formacie YAML lub JSON. Spójrz na poniższy przykład:

```
apiVersion: beta.kuar.com/v1
kind: LoadTest
metadata:
  name: my-loadtest
spec:
  service: my-service
  scheme: https
  requestsPerSecond: 1000
  paths:
  - /index.html
  - /login.html
  - /shares/my-shares/
```

Zwróć uwagę, że w obiekcie `CustomResourceDefinition` naszego zasobu brak definicji schematu. Wprawdzie można zdefiniować specyfikację OpenAPI, ale w przypadku prostych typów zasobów ta gra nie jest warta świeczki. Jeśli potrzebna jest możliwość walidacji, można zarejestrować kontroler wstępu w sposób opisany w dalszej części tego rozdziału.

Teraz naszego pliku *loadtest.yaml* możemy użyć do utworzenia zasobu w taki sam sposób, w jaki tworzy się zasoby typów wbudowanych:

```
$ kubectl create -f loadtest.yaml
```

Aktualnie na liście zasobów `loadtests` pojawi się nasz nowo utworzony zasób:

```
$ kubectl get loadtests
```

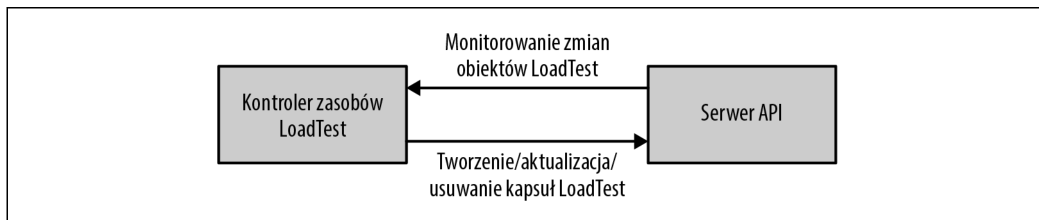
Choć udało nam się już coś osiągnąć, na razie w naszym przykładzie nie dzieje się nic ciekawego. Oczywiście możemy użyć tego prostego API CRUD (tworzenie, odczyt, aktualizacja, usuwanie) do pracy z danymi obiektów `LoadTest`, ale na razie nie są tworzone żadne testy obciążeniowe w reakcji na ten nowy interfejs API.

Przyczyną tego jest brak w klastrze kontrolera reagującego w odpowiedni sposób na pojawienie się definicji obiektu `LoadTest`. Zasób `LoadTest` jest tylko połową infrastruktury potrzebnej do dodawania obiektów `LoadTest` do naszego klastra. Druga połowa to kod, który będzie nieustannie monitorował zasoby oraz tworzył, modyfikował lub usuwał obiekty `LoadTest` według potrzeby.

Kontroler, jak przystało na użytkownika API, pobiera od serwera API listę obiektów `LoadTest` i obserwuje, czy zachodzą jakieś zmiany. Na rysunku 16.2 pokazaliśmy, jak odbywa się ta interakcja między kontrolerem i serwerem API.

Kod takiego kontrolera może być zarówno bardzo prosty, jak i skomplikowany. Najprostsze kontrolery zawierają pętlę `for`, za pomocą której nieustannie sprawdzają, czy pojawiły się nowe obiekty, i tworzą lub usuwają implementujące je zasoby (na przykład kapsuły robocze `LoadTest`).

Kup ksi k Pole ksi k



Rysunek 16.2. Interakcje zasobu CustomResourceDefinition

Niestety to rozwiązanie oparte na ciągłym sprawdzaniu jest nieefektywne, ponieważ pętla stanowi dodatkowe obciążenie dla serwera API. Lepszym wyjściem jest użycie obserwacyjnego interfejsu API na serwerze API, który wysyła strumień aktualizacji, gdy się pojawią, eliminując w ten sposób zarówno opóźnienie, jak i narzut związane z ciągłym sprawdzaniem. Tylko że trudno jest bezbłędnie posłużyć się tym interfejsem API. Jeśli więc chcesz używać czujek, najlepiej skorzystaj z dobrze zaprojektowanego mechanizmu, takiego jak na przykład wzorzec Informer udostępniany w bibliotece `client-go` (*https://godoc.org/k8s.io/client-go/informers*).

Po utworzeniu własnego zasobu i zaimplementowaniu go przez kontroler w naszym klastrze możemy korzystać z nowej funkcji. Trzeba jednak przyznać, że nadal brakuje nam pewnych mechanizmów, które powinien mieć każdy dobrze działający zasób. Dwa najważniejsze z nich to walidacja i wybór wartości domyślnych. Walidacja to proces sprawdzania, czy obiekty `LoadTest` wysyłane do serwera API są poprawne i nadają się do tworzenia testów, natomiast wybór wartości domyślnych ułatwia korzystanie z naszych zasobów przez automatyczne dostarczanie typowych wartości domyślnych. Dodamy teraz te dwa składniki funkcjonalności do naszego zasobu.

Jak napisaliśmy wcześniej, mechanizm walidacji naszych obiektów możemy zdefiniować w postaci specyfikacji OpenAPI. Ta metoda sprawdzi się w przypadku podstawowych testów obecności wymaganych pól lub nieobecności nieznanych pól. Opis OpenAPI wykracza poza zakres tej książki, ale w internecie można znaleźć wiele materiałów na ten temat. Jednym z cennych źródeł jest pełna specyfikacja API Kubernetes (*http://bit.ly/2Jn89we*).

Generalnie rzecz biorąc, schemat API nie wystarczy do walidacji obiektów API. Na przykład w `loadtests` możemy chcieć sprawdzać, czy obiekt `LoadTest` ma prawidłowy schemat (przykładowo `http` lub `https`) lub czy wartość `requestsPerSecond` jest większa od zera.

W celu zaimplementowania takich testów użyjemy walidacyjnego kontrolera wstępu. Jak napisaliśmy wcześniej, kontrolery te przechwytują żądania do serwera API, zanim te żądania zostaną przetworzone, i mogą je odrzucać lub modyfikować w locie. Kontrolery wstępu można dodawać do klastrów przez dynamiczny system kontroli wstępu. Dynamiczny kontroler wstępu to prosta aplikacja HTTP. Serwer API łączy się z nim przez obiekt usługi Kubernetes lub dowolny adres URL. Oznacza to, że kontrolery wstępu mogą działać poza klastrem — na przykład jako funkcja usługowa w ramach oferty dostawcy chmury, jak Azure Functions czy AWS Lambda.

Aby zainstalować walidacyjny kontroler wstępu, należy go zdefiniować jako obiekt Kubernetes `ValidatingWebhookConfiguration`. Obiekt ten określa miejsce działania kontrolera oraz zasób (w tym przypadku `LoadTest`) i akcję (w tym przypadku `CREATE`), w której kontroler ten ma działać. Poniżej znajduje się kompletna definicja walidacyjnego kontrolera wstępu:

Kup ksi k   Pole ksi k

```
apiVersion: admissionregistration.k8s.io/v1beta1
kind: ValidatingWebhookConfiguration
metadata:
  name: kuar-validator
webhooks:
  - name: validator.kuar.com
    rules:
    - apiGroups:
      - "beta.kuar.com"
      apiVersions:
      - v1
      - CREATE
    resources:
      - loadtests
    clientConfig:
      # Wpisz adres IP swojego elementu webhook
      url: https://192.168.1.233:8080
      # To powinien być certyfikat Twojego klastra w formacie base64
      # Znajduje się on w pliku ${KUBECONFIG}
      caBundle: ZAMIEŃMNIE
```

Na szczęście pod względem bezpieczeństwa i na nieszczęście pod kątem poziomu złożoności serwer API Kubernetes dostęp do elementów webhook może uzyskiwać tylko przez HTTPS. To znaczy, że musimy wygenerować certyfikat do obsługi elementu webhook. Najprostszym sposobem jest użycie funkcji generowania certyfikatów za pomocą własnej organizacji certyfikacyjnej klastra.

Najpierw potrzebujemy klucza prywatnego i żądania podpisania certyfikatu (CSR). Oto prosty program w języku Go, który je generuje:

```
package main

import (
        "crypto/rand"
        "crypto/rsa"
        "crypto/x509"
        "crypto/x509/pkix"
        "encoding/asn1"
        "encoding/pem"
        "net/url"
        "os"
)

func main() {
        host := os.Args[1]
        name := "server"

        key, err := rsa.GenerateKey(rand.Reader, 1024)
        if err != nil {
                panic(err)
        }
        keyDer := x509.MarshalPKCS1PrivateKey(key)
        keyBlock := pem.Block{
                Type: "KLUCZ PRYWATNY RSA",
                Bytes: keyDer,
        }
        keyFile, err := os.Create(name + ".key")
        if err != nil {
```

```
                panic(err)
        }
        pem.Encode(keyFile, &keyBlock)
        keyFile.Close()
        commonName := "myuser"
        emailAddress := "someone@myco.com"

        org := "My Co, Inc."
        orgUnit := "Widget Farmers"
        city := "Seattle"
        state := "WA"
        country := "US"

        subject := pkix.Name{
                CommonName: commonName,
                Country: []string{country},
                Locality: []string{city},
                Organization: []string{org},
                OrganizationalUnit: []string{orgUnit},
                Province: []string{state},
        }

        uri, err := url.ParseRequestURI(host)
        if err != nil {
                panic(err)
        }

        asn1, err := asn1.Marshal(subject.ToRDNSequence())
        if err != nil {
                panic(err)
        }
        csr := x509.CertificateRequest{
                RawSubject: asn1,
                EmailAddresses: []string{emailAddress},
                SignatureAlgorithm: x509.SHA256WithRSA,
                URIs: []*url.URL{uri},
        }
        bytes, err := x509.CreateCertificateRequest(rand.Reader, &csr, key)
        if err != nil {
                panic(err)
        }
        csrFile, err := os.Create(name + ".csr")
        if err != nil {
                panic(err)
        }
        pem.Encode(csrFile, &pem.Block{Type: "CERTIFICATE REQUEST", Bytes: bytes})
        csrFile.Close()
}
```

Ten program można uruchomić za pomocą następującego polecenia:

```
$ go run csr-gen.go <URL-elementu-webook>
```

Wygeneruje on dwa pliki — *server.csr* i *server-key.pem*.

Następnie możemy utworzyć żądanie podpisania certyfikatu do serwera API Kubernetes przy użyciu przedstawionego kodu YAML:

```
apiVersion: certificates.k8s.io/v1beta1
```


Kup ksi k Pole ksi k

```
kind: CertificateSigningRequest
metadata:
  name: validating-controller.default
spec:
  groups:
  - system:authenticated
  request: ZAMIEŃMNIE
  usages:
  usages:
  - digital signature
  - key encipherment
  - key agreement
  - server auth
```

W polu `request` znajduje się wartość `ZAMIEŃMNIE`, którą należy zamienić na żądanie podpisania certyfikatu w formacie base64 wygenerowane w poprzednim kodzie:

```
$ perl -pi -e s/REPLACEME/$(base64 server.csr | tr -d '\n')/ \
admission-controller-csr.yaml
```

Gdy ma się gotowe żądanie podpisania certyfikatu, można wysłać je do serwera API, aby uzyskać certyfikat:

```
$ kubectl create -f admission-controller-csr.yaml
```

Następnie należy zatwierdzić żądanie:

```
$ kubectl certificate approve validating-controller.default
```

Po zatwierdzeniu można pobrać nowy certyfikat:

```
$ kubectl get csr validating-controller.default -o json | \
jq -r .status.certificate | base64 -d > server.crt
```

Jeśli masz certyfikat, możesz w końcu utworzyć kontroler wstępu oparty na SSL (uf!). Gdy kontroler ten otrzyma żądanie, zawiera obiekt typu `AdmissionReview`, w którym znajdują się metadane dotyczące żądania, jak również samą treść właściwą. Nasz walidacyjny kontroler wstępu zarejestrowaliśmy tylko dla jednego typu zasobów i jednej akcji (`CREATE`), więc nie musimy analizować metadanych żądania. Zamiast tego od razu zagłębiamy się w zasób i sprawdzamy, czy wartość `requestsPerSecond` jest dodatnia oraz czy schemat adresu URL jest poprawny. Jeśli coś jest nie tak, zwracamy odpowiedź odmowną w formacie JSON.

Implementacja domyślnych ustawień w kontrolerze wstępu jest podobna do poprzedniej implementacji, tylko zamiast konfiguracji `ValidatingWebhookConfiguration` użyjemy `MutatingWebhookConfiguration` i dostarczymy obiekt `JSONPatch` w celu modyfikacji obiektu żądania przed jego zapisaniem.

Poniżej znajduje się fragment kodu w języku TypeScript, który możesz dodać do swojego kontrolera, aby ustawiał wartości domyślne. Jeśli pole `paths` w `loadtest` ma zerową długość, dodajemy jedną ścieżkę do */index.html*:

```
if (needsPatch(loadtest)) {
    const patch = [
        { 'op': 'add', 'path': '/spec/paths', 'value': ['/index.html'] },
    ]
    response['patch'] = Buffer.from(JSON.stringify(patch))
```


Kup ksi k   Pole ksi k

```
            .toString('base64');
        response['patchType'] = 'JSONPatch';
    }
```

Następnie możemy zarejestrować ten element webhook jako konfigurację `MutatingWebhookConfiguration`, zmieniając pole `kind` w obiekcie YAML i zapisując plik jako *mutatingcontroller.yaml*. W dalszej kolejności tworzymy kontroler, wykonując poniższe polecenie:

```
$ kubectl create -f mutating-controller.yaml
```

W ten sposób stworzyliśmy kompletne rozszerzenie serwera API Kubernetes na bazie własnych zasobów i kontrolerów wstępu. W następnym podrozdziale opisujemy kilka ogólnych wzorców tworzenia rozszerzeń.

Wzorce tworzenia zasobów

Nie wszystkie własne zasoby użytkownika są takie same. Interfejs API Kubernetes rozszerza się w różnych celach, dlatego poniżej przedstawiamy kilka ogólnych wzorców, które warto znać.

Tylko dane

Najprostszy wzorzec rozszerzania API to „tylko dane". Polega on na wykorzystywaniu serwera API wyłącznie do zapisywania i pobierania danych dla aplikacji. Należy podkreślić, że serwera API Kubernetes **nie** powinno się używać do przechowywania danych aplikacji, ponieważ nie jest to magazyn par klucz-wartość. Zamiast tego należy używać rozszerzeń będących obiektami kontroli lub konfiguracji pomagającymi we wdrażaniu albo wykonywaniu aplikacji. Przykładowym zastosowaniem wzorca „tylko dane" jest konfiguracja wdrożeń canary aplikacji — na przykład kierowanie 10% całości ruchu do eksperymentalnego backendu. Teoretycznie takie informacje konfiguracyjne można przechowywać w obiekcie `ConfigMap`, ale obiekty te nie są typizowane, przez co w wielu przypadkach lepszym i prostszym rozwiązaniem jest użycie ściślej typizowanego obiektu rozszerzenia API.

Rozszerzenia reprezentujące tylko dane nie wymagają kontrolera do aktywacji, ale ewentualnie można zdefiniować kontroler walidacyjny lub mutacyjny w celu zapewnienia poprawności. Na przykład w przypadku canary kontroler walidacyjny mógłby sprawdzać, czy suma wartości procentowych wynosi 100%.

Kompilatory

Wzorzec „kompilator" lub „abstrakcja" jest odrobinę bardziej skomplikowany. W jego przypadku obiekt rozszerzenia API reprezentuje abstrakcję wyższego poziomu, która jest „wkompilowana" w kombinację obiektów Kubernetes niższego poziomu. Przykładem zastosowania tego wzorca w praktyce jest rozszerzenie `LoadTest` z poprzedniego przykładu. Użytkownik używa rozszerzenia jako koncepcji wysokiego poziomu, w tym przypadku `loadtest`, ale koncepcja ta powstaje poprzez wdrożenie jako kolekcji kapsuł i usług Kubernetes. Aby to osiągnąć, gdzieś w klastrze musi działać kontroler API obserwujący bieżące obiekty `LoadTest` i tworzący „skompilowaną" reprezentację (oraz usuwający te obiekty, które już nie istnieją). Inaczej niż w przypadku wzorca „operator", który jest opisany w następnym punkcie, w skompilowanych abstrakcjach nie ma mechanizmu sprawdzania stanu. Czynność ta zostaje oddelegowana do obiektów niższego poziomu (na przykład kapsuł).

Operatory

Podczas gdy rozszerzenia według wzorca „kompilator” dostarczają łatwych w użyciu abstrakcji, rozszerzenia typu „operator” umożliwiają aktywne zarządzanie zasobami tworzonymi przez rozszerzenia. Mogą one dostarczać abstrakcje wyższego poziomu (na przykład bazę danych), które są kompilowane do postaci reprezentacji niższego poziomu, ale dodatkowo zapewniają funkcjonalność online, na przykład wykonywanie migawek bazy danych czy wysyłanie powiadomień o dostępności nowych uaktualnień. To wymaga, aby kontroler nie tylko monitorował API rozszerzeń w celu dodawania lub usuwania pewnych elementów, ale również by monitorował informacje o stanie działania aplikacji przekazywane przez rozszerzenie (na przykład bazę danych) i podejmował właściwe działania zaradcze w przypadku problemów z bazami danych, wykonywał migawki lub przywracał je w razie awarii. „Operator” to najbardziej skomplikowany, a jednocześnie dający największe możliwości wzorzec rozszerzeń API Kubernetes. Umożliwia on użytkownikowi tworzenie abstrakcji, które nie tylko wspomagają wdrożenia, ale dodatkowo potrafią sprawdzać stan i przeprowadzać naprawy.

Jak zacząć

Początki pracy z rozszerzeniami API Kubernetes mogą być nużące i wyczerpujące, ale na szczęście istnieje bardzo dużo pomocnego kodu, który można wykorzystać. Projekt Kubebuilder (*https://kubebuilder.io/*) zawiera bibliotekę kodu, za pomocą którego można z łatwością tworzyć solidne rozszerzenia API Kubernetes. To doskonałe źródło na początek dla każdego, kto chce nauczyć się tworzyć rozszerzenia.

Podsumowanie

Jedną z największych zalet systemu Kubernetes jest jego ekosystem, który prosperuje między innymi dzięki doskonałym możliwościom rozszerzania API Kubernetes. Nieważne, czy projektujesz własne rozszerzenia w celu modyfikacji sposobu działania klastra, czy korzystasz z gotowych rozszerzeń jako narzędzi, usług klastrowych lub operatorów, rozszerzenia API są podstawowym rozwiązaniem do budowy indywidualnie dostosowanych klastrów i środowiska odpowiedniego do szybkiego tworzenia niezawodnych aplikacji.

Kup ksi k

Pole ksi k

PROGRAM PARTNERSKI

GRUPY HELION

1. ZAREJESTRUJ SIĘ
2. PREZENTUJ KSIĄŻKI
3. ZBIERAJ PROWIZJĘ

Zmień swoją stronę WWW w działający bankomat!

Dowiedz się więcej i dołącz już dzisiaj!
http://program-partnerski.helion.pl

Kubernetes: systemy rozproszone mogą być skalowalne i niezawodne!

Kubernetes jest czymś więcej niż platformą do orkiestracji kontenerów. W ciągu zaledwie kilku lat stał się najpopularniejszym i najbogatszym narzędziem do tworzenia, wdrażania i utrzymywania aplikacji w chmurze. Tak radykalna zmiana sposobu funkcjonowania systemów informatycznych wymaga przemodelowania podejścia i stylu pracy zespołów programistycznych. Jeśli jednak wypróbujesz Kubernetesa, przekonasz się, że bardzo upraszcza on tworzenie, wdrażanie i utrzymywanie systemów rozproszonych.

Ta książka jest przeznaczona dla początkujących i zaawansowanych użytkowników Kubernetesa. Opisano tu, jak działa orkiestrator klastrów Kubernetes oraz jak wykorzystać jego narzędzia i interfejsy API do usprawnienia procesów rozwoju, dostarczania i utrzymywania rozproszonych aplikacji. Wyjaśniono niezbędne szczegóły dotyczące aplikacji kontenerowych, uruchamiania i obsługi klastrów oraz wdrażania aplikacji w Kubernetesie. Przedstawiono sposoby integracji magazynów danych i rozszerzania platformy. Przydatnym podsumowaniem treści zawartych w książce jest kilka praktycznych przykładów tworzenia i wdrażania rzeczywistych aplikacji w Kubernetesie, a także omówienie metod organizowania aplikacji w systemie kontroli źródła.

Najciekawsze zagadnienia:

- tworzenie i uruchamianie klastrów Kubernetesa
- projektowanie aplikacji: kapsuły, usługi, narzędzia Ingress i obiekty ReplicaSet
- integracja magazynów danych z kontenerowymi mikrousługami
- obiekty specjalne: DaemonSet, Job, ConfigMap i tajne dane
- praktyczne przykłady tworzenia i wdrażania rzeczywistych aplikacji w Kubernetes

Brendan Burns jest wybitnym inżynierem w Microsoft Azure, gdzie kieruje zespołami zajmującymi się technologiami DevOps, open source i mikrousługami. Współtworzył projekt Kubernetes w firmie Google.
Joe Beda jest dyrektorem technicznym w firmie Vmware. Brał udział w tworzeniu projektu Kubernetes. Obecnie wdraża strategię jego rozwoju w firmie VMware.
Kelsey Hightower jest specjalistą do spraw rozwoju kadr dla Google Cloud Platform. Ma duże doświadczenie jako lider zespołów zajmujących się dostarczaniem oprogramowania.

Helion
helion.pl
HELION SA
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl

Sprawdź nasze szkolenia!
SZKOLENIA
AKADEMIA IT & BUSINESS
HELIONSZKOLENIA.PL
KOD KORZYŚCI
Sięgnij po więcej!

ISBN 978-83-283-6745-6

INFORMATYKA W NAJLEPSZYM WYDANIU
Cena: 59,00 zł