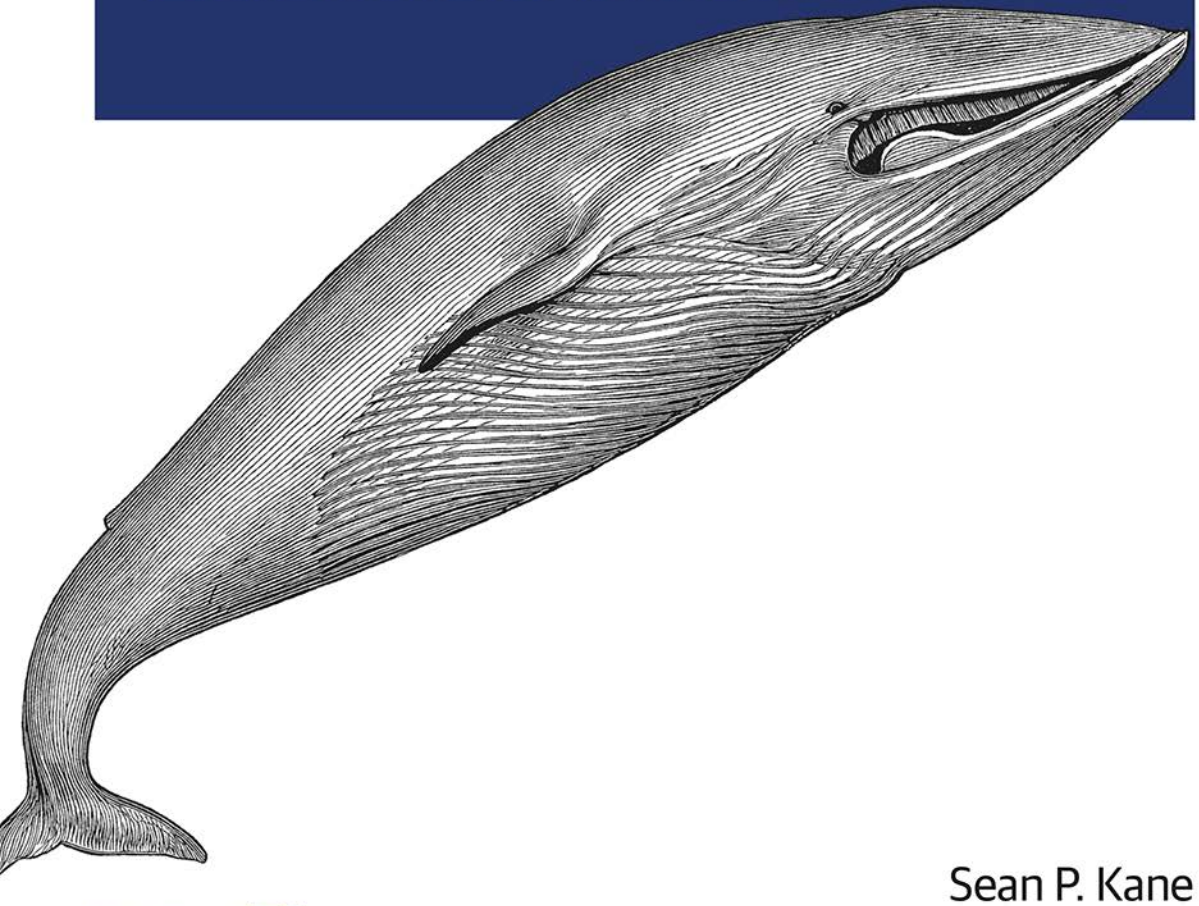
O'REILLY®

Wydanie II

Docker

Praktyczne zastosowania



Sean P. Kane
Karl Matthias

Helion

Tytuł oryginału: Docker: Up & Running: Shipping Reliable Containers in Production, 2nd Edition

Tłumaczenie: Andrzej Stefański

ISBN: 978-83-283-5604-7

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)

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/docke2
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

Spis treści

ROZDZIAŁ 8.
Docker Compose

W tym momencie powinieneś dobrze już znać polecenie `docker` i wiedzieć, jak można go użyć do budowania, uruchamiania, monitorowania i debugowania swoich aplikacji. Skoro już czujesz się komfortowo, pracując z pojedynczymi kontenerami, nie upłynie wiele czasu i będziesz chciał dzielić się swoimi projektami i zacząć budować bardziej złożone projekty, wymagające wielu kontenerów do poprawnego działania. Szczególnie może przydać się to w środowiskach deweloperskich, gdzie uruchamiając zestaw kontenerów, można łatwo zasymulować wiele środowisk produkcyjnych na swojej lokalnej maszynie.

Jeśli jednak uruchamiasz cały stos kontenerów, każdy kontener musi zostać uruchomiony w odpowiedni sposób, aby sprawić, że działająca w nim aplikacja jest poprawnie skonfigurowana i będzie działała zgodnie z oczekiwaniami. Dbanie za każdym razem o poprawność tych ustawień może być wyzwaniem, szczególnie jeśli nie jesteś osobą, która napisała aplikację. Aby sobie w tym pomóc, podczas tworzenia oprogramowania ludzie często piszą skrypty powłoki, które mogą budować i uruchamiać ich kontenery w spójny sposób. Choć to działa, może stać się trudne do zrozumienia dla kolejnych osób i trudne w utrzymaniu, gdy projekt będzie zmieniał się w czasie. Nie musi to też być powtarzalne w różnych projektach.

Aby pomóc w rozwiązaniu tego problemu, firma Docker Inc. wydała narzędzie **Docker Compose**, początkowo przeznaczone tylko dla deweloperów. Narzędzie to jest dołączone do Docker Community Edition, ale możesz również zainstalować je, korzystając z instrukcji opublikowanej na stronie internetowej (*https://docs.docker.com/compose/install*).

Docker Compose to niesamowicie użyteczne narzędzie, które może uporządkować różnego rodzaju zadania związane z tworzeniem oprogramowania, zazwyczaj uciążliwe i podatne na błędy. Może być łatwo wykorzystane, by wspomóc deweloperów przy szybkim tworzeniu skomplikowanych stosów aplikacji, kompilowania aplikacji bez potrzeby lokalnego tworzenia złożonych środowisk programistycznych i w wielu innych zadaniach.

W tym rozdziale przyjrzymy się, w jaki sposób można najlepiej wykorzystać Compose. Będziemy korzystali z repozytorium GitHub we wszystkich dalszych przykładach. Jeśli chcesz uruchamiać przykłady, podczas czytania tekstu powinieneś uruchomić poniższe polecenie, aby pobrać kod (jeżeli nie zrobiłeś tego już w rozdziale 6.).

```
$ git clone https://github.com/spkane/rocketchat-hubot-demo.git --config core.autocrlf=input
```

W przykładowym skrypcie powłoki i pliku *docker-compose.yaml* pokazanych poniżej niektóre wiersze zostały skrócone, by dopasować je do marginesów. Upewnij się, że korzystasz z plików pobranych z wskazanego wyżej repozytorium git, jeśli planujesz samodzielnie uruchamiać te przykłady.

Repozytorium to zawiera konfigurację, jakiej będziemy potrzebowali, by uruchomić kompletny serwis web, który składa się z bazy danych *MongoDB*, serwera do komunikacji *RocketChat* z otwartymi źródłami, bota *Hubot* (*https://goo.gl/hKT3QW*) oraz instancji *zmachine-api*, by dodać do tego odrobinę zabawy.

Konfigurowanie Docker Compose

Zanim zagłębimy się w techniki stosowania polecenia `docker-compose`, warto przyjrzeć się, jakiego rodzaju narzędzia *ad hoc* ono zastępuje. Poświęcimy więc chwilę, by przyjrzeć się skryptowi powłoki, jaki mógłby zostać użyty do zbudowania i uruchomienia lokalnej kopii naszego serwisu wykorzystywanej przy programowaniu i do lokalnego testowania za pomocą Dockera. Jest to długi i szczegółowy skrypt, ale jest to ważne, by uzasadnić stwierdzenie, że Docker Compose stanowi duży skok do przodu w stosunku do skryptów powłoki.

```
#!/bin/bash

set -e
set -u

if [ $# -ne 0 ] && [ ${1} == "down" ]; then
  docker rm -f hubot || true
  docker rm -f zmachine || true
  docker rm -f rocketchat || true
  docker rm -f mongo-init-replica || true
  docker rm -f mongo || true
  docker network rm botnet || true
  echo "Environment torn down..."
  exit 0
fi

# Global Settings
export PORT="3000"
export ROOT_URL="http://127.0.0.1:3000"
export MONGO_URL="mongodb://mongo:27017/rocketchat"
export MONGO_OPLOG_URL="mongodb://mongo:27017/local"
export MAIL_URL="smtp://smtp.email"
export RESPOND_TO_DM="true"
export HUBOT_ALIAS=". "
export LISTEN_ON_ALL_PUBLIC="true"
export ROCKETCHAT_AUTH="password"
export ROCKETCHAT_URL="rocketchat:3000"
export ROCKETCHAT_ROOM=""
export ROCKETCHAT_USER="hubot"
export ROCKETCHAT_PASSWORD="HughTheBot"
export BOT_NAME="bot"
export EXTERNAL_SCRIPTS="hubot-help,hubot-diagnostics,hubot-zmachine"
export HUBOT_ZMACHINE_SERVER="http://zmachine:80"
export HUBOT_ZMACHINE_ROOMS="zmachine"
```

```
export HUBOT_ZMACHINE_OT_PREFIX="ot"

docker build -t spkane/mongo:3.2 ./mongodb/docker

docker push spkane/mongo:3.2
docker pull spkane/zmachine-api:latest
docker pull rocketchat/rocket.chat:0.61.0
docker pull rocketchat/hubot-rocketchat:latest

docker rm -f hubot || true
docker rm -f zmachine || true
docker rm -f rocketchat || true
docker rm -f mongo-init-replica || true
docker rm -f mongo || true

docker network rm botnet || true

docker network create -d bridge botnet

docker run -d \
  --name=mongo \
  --network=botnet \
  --restart unless-stopped \
  -v $(pwd)/mongodb/data/db:/data/db \
  spkane/mongo:3.2 \
  mongod --smallfiles --oplogSize 128 --replSet rs0
sleep 5
docker run -d \
  --name=mongo-init-replica \
  --network=botnet \
  spkane/mongo:3.2 \
  'mongo mongo/rocketchat --eval "rs.initiate({ ..."'
sleep 5
docker run -d \
  --name=rocketchat \
  --network=botnet \
  --restart unless-stopped \
  -v $(pwd)/rocketchat/data/uploads:/app/uploads \
  -p 3000:3000 \
  -e PORT=${PORT} \
  -e ROOT_URL=${ROOT_URL} \
  -e MONGO_URL=${MONGO_URL} \
  -e MONGO_OPLOG_URL=${MONGO_OPLOG_URL} \
  -e MAIL_URL=${MAIL_URL} \
  rocketchat/rocket.chat:0.61.0
docker run -d \
  --name=zmachine \
  --network=botnet \
  --restart unless-stopped \
  -v $(pwd)/zmachine/saves:/root/saves \
  -v $(pwd)/zmachine/zcode:/root/zcode \
  -p 3002:80 \
  spkane/zmachine-api:latest
docker run -d \
  --name=hubot \
  --network=botnet \
  --restart unless-stopped \
  -v $(pwd)/hubot/scripts:/home/hubot/scripts \
```

```
  -p 3001:8080 \
  -e RESPOND_TO_DM="true" \
  -e HUBOT_ALIAS=". " \
  -e LISTEN_ON_ALL_PUBLIC="true" \
  -e ROCKETCHAT_AUTH="password" \
  -e ROCKETCHAT_URL="rocketchat:3000" \
  -e ROCKETCHAT_ROOM="" \
  -e ROCKETCHAT_USER="hubot" \
  -e ROCKETCHAT_PASSWORD="HughTheBot" \
  -e BOT_NAME="bot" \
  -e EXTERNAL_SCRIPTS="hubot-help,hubot-diagnostics,hubot-zmachine" \
  -e HUBOT_ZMACHINE_SERVER="http://zmachine:80" \
  -e HUBOT_ZMACHINE_ROOMS="zmachine" \
  -e HUBOT_ZMACHINE_OT_PREFIX="ot" \
  rocketchat/hubot-rocketchat:latest
echo "Environment setup..."
exit 0
```

W tej chwili prawdopodobnie jesteś w stanie dość łatwo odczytać większość tego skryptu. Jak może już zauważyłeś, jest on dość trudny do odczytania, niezbyt elastyczny, zapewne bardzo trudny do edycji i w kilku miejscach może zachować się w nieprzewidywalny sposób. Jeśli chcielibyśmy przestrzegać dobrych praktyk dla skryptów powłoki i obsłużyć w nim wszystkie możliwe błędy, by doprowadzić do jego powtarzalności, byłby też dwa do trzech razy dłuższy, niż jest teraz. Bez włożenia dużej ilości pracy w wyizolowanie wspólnych funkcjonalności służących do obsługi błędów musiałbyś też przepisywać większość tej logiki za każdym razem przy tworzeniu nowego projektu tego typu. Nie jest to zbyt dobry sposób na przygotowanie procesu, który musi działać za każdym razem. To tutaj wkraczają do akcji dobre narzędzia. Korzystając z Docker Compose, możesz osiągnąć dokładnie ten sam cel, sprawiając jednocześnie, że będzie to dużo bardziej powtarzalne i łatwiejsze do przeczytania, zrozumienia i utrzymania.

W odróżnieniu od tego zagmatwanego skryptu powłoki, który zawiera wiele powtórzeń i jest podatny na błędy, Docker Compose jest zazwyczaj konfigurowany za pomocą jednego pliku YAML (*http://yaml.org/*) z deklaracjami dla każdego projektu o nazwie *docker-compose.yaml*. Ten plik konfiguracyjny bardzo łatwo się czyta i będzie działał w bardzo powtarzalny sposób, dzięki czemu każdy użytkownik uzyska ten sam efekt, uruchamiając go. Poniżej możesz zobaczyć przykładowy plik *docker-compose.yaml*, który może zastąpić wcześniej zaprezentowany skrypt powłoki:

```
version: '2'
services:
  mongo:
    build:
      context: ../../mongodb/docker
    image: spkane/mongo:3.2
    restart: unless-stopped
    command: mongod --smallfiles --oplogSize 128 --replSet rs0
    volumes:
      - "../../mongodb/data/db:/data/db"
    networks:
      - botnet
  mongo-init-replica:
    image: spkane/mongo:3.2
    command: 'mongo mongo/rocketchat --eval "rs.initiate({ ..."'
    depends_on:
      - mongo
```

```
    networks:
      - botnet
  rocketchat:
    image: rocketchat/rocket.chat:0.61.0
    restart: unless-stopped
    volumes:
      - "../../rocketchat/data/uploads:/app/uploads"
    environment:
      PORT: 3000
      ROOT_URL: "http://127.0.0.1:3000"
      MONGO_URL: "mongodb://mongo:27017/rocketchat"
      MONGO_OPLOG_URL: "mongodb://mongo:27017/local"
      MAIL_URL: "smtp://smtp.email"
    depends_on:
      - mongo
    ports:
      - 3000:3000
    networks:
      - botnet
  zmachine:
    image: spkane/zmachine-api:latest
    restart: unless-stopped
    volumes:
      - "../../zmachine/saves:/root/saves"
      - "../../zmachine/zcode:/root/zcode"
    depends_on:
      - rocketchat
    expose:
      - "80"
    networks:
      - botnet
  hubot:
    image: rocketchat/hubot-rocketchat:latest
    restart: unless-stopped
    volumes:
      - "../../hubot/scripts:/home/hubot/scripts"
    environment:
      RESPOND_TO_DM: "true"
      HUBOT_ALIAS: ". "
      LISTEN_ON_ALL_PUBLIC: "true"
      ROCKETCHAT_AUTH: "password"
      ROCKETCHAT_URL: "rocketchat:3000"
      ROCKETCHAT_ROOM: ""
      ROCKETCHAT_USER: "hubot"
      ROCKETCHAT_PASSWORD: "HughTheBot"
      BOT_NAME: "bot"
      EXTERNAL_SCRIPTS: "hubot-help,hubot-diagnostics,hubot-zmachine"
      HUBOT_ZMACHINE_SERVER: "http://zmachine:80"
      HUBOT_ZMACHINE_ROOMS: "zmachine"
      HUBOT_ZMACHINE_OT_PREFIX: "ot"
    depends_on:
      - zmachine
    ports:
      - 3001:8080
    networks:
      - botnet
networks:
  botnet:
    driver: bridge
```

Plik *docker-compose.yaml* ułatwia opisanie wszystkich ważnych wymagań dla każdego z Twoich serwisów oraz sposobów komunikacji pomiędzy nimi. Otrzymujemy też za darmo dużo mechanizmów sprawdzania składni i logiki, których nie mieliśmy nawet czasu napisać w naszym skrypcie powłoki, a które i tak czasem okazałyby się niepoprawne, niezależnie od tego, jak ostrożni byśmy byli.

A więc co kazaliśmy Compose zrobić w tym pliku YAML? Pierwszy wiersz naszego pliku po prostu mówi Docker Compose, w której wersji języka konfiguracji Compose (*https://docs.docker.com/compose/compose-le*) ten plik został zapisany.

```
version: '2'
```

Reszta naszego dokumentu podzielona jest na dwie sekcje: `services` i `networks`.

Na początek zerknijmy na sekcję `networks`. W tym pliku *docker-compose.yaml* definiujemy jedną nazwaną sieć Dockera.

```
networks:
  botnet:
    driver: bridge
```

Jest to bardzo prosta konfiguracja, nakazująca Docker Compose utworzyć jedną sieć o nazwie `botnet`, korzystającą z (domyślnego) sterownika `bridge` (ang. *most*), który połączy sieć Dockera ze stosem sieciowym hosta.

Sekcja `services` jest najważniejszą częścią konfiguracji i mówi Docker Compose, jakie aplikacje chcesz uruchomić. Tutaj sekcja `services` definiuje pięć serwisów: `mongo`, `mongo-init-replica`, `rocketchat`, `zmachine` i `hubot`. Następnie każdy nazwany serwis zawiera sekcje, które mówią Dockerowi, w jaki sposób zbudować, skonfigurować i uruchomić dany serwis.

Jeśli przyjrzysz się serwisowi `mongo`, zobaczysz, że pierwsza sekcja nosi nazwę `build` i zawiera klucz `context`. Jest to informacja dla Docker Compose, że może zbudować ten obraz i że pliki potrzebne do jego zbudowania są umieszczone w katalogu *../../mongodb/docker*, który znajduje się dwa poziomy powyżej katalogu zawierającego plik *docker-compose.yaml*.

```
build:
  context: ../../mongodb/docker
```

Następne ustawienie, `image`, definiuje znacznik obrazu, jaki chcesz zastosować przy budowaniu lub pobieraniu (jeśli nie budujesz obrazu), a następnie uruchomić.

```
image: spkane/mongo:3.2
```

Za pomocą opcji `restart` mówisz Dockerowi, kiedy ma restartować Twoje kontenery. W większości przypadków będziesz chciał, by Docker restartował kontenery za każdym razem, gdy nie zostaną przez Ciebie jawnie zatrzymane.

```
restart: unless-stopped
```

Opcja `command` umożliwia Ci definiowanie polecenia, które Twój kontener powinien uruchamiać podczas startu. Dla tej usługi uruchomimy MongoDB z kilkoma parametrami przekazywanymi w wierszu poleceń.

```
command: mongod --smallfiles --oplogSize 128 --replSet rs0
```

Wiele usług ma jakieś dane, które muszą być zapisywane podczas tworzenia, mimo efemerycznej natury kontenerów. Aby to obsłużyć, najłatwiej zamontować w kontenerach lokalny katalog. Sekcja `volumes` umożliwia Ci wypisanie wszystkich lokalnych katalogów, które chciałbyś zamontować w kontenerze i zdefiniować, gdzie się one pojawią.

```
volumes:
  - "../../mongodb/data/db:/data/db"
```

Ostatnia podsekcja definicji usługi `mongo`, `networks`, mówi programowi Docker Compose, do których sieci ten kontener powinien być podłączony.

```
networks:
  - botnet
```

W tym momencie przeskoczmy do usługi `rocketchat`. Ta usługa nie ma podsekcji `build`. Definiuje ona jedynie obraz bez wiersza `build`, który mówi Docker Compose, że nie może zbudować tego obrazu i musi zamiast tego spróbować pobrać i uruchomić standardowy istniejący obraz Dockera z danym znacznikiem.

Pierwszą nowością, jaką możesz zauważyć w tym serwisie, jest nazwa `environment`. Jest to miejsce, w którym możesz zdefiniować dowolne zmienne środowiska, jakie zechcesz przekazać do Twojego kontenera.

Zauważ, że wartość zmiennej `MONGO_URL` nie zawiera adresu IP czy pełnej nazwy domenowej (FQDN). Jest tak dlatego, że wszystkie te usługi uruchomione są w tej samej sieci Dockera i Docker konfiguruje każdy kontener w taki sposób, że może on odnaleźć inne za pomocą nazwy usługi. Oznacza to, że możemy łatwo tworzyć adresy URL takie jak ten, by w prosty sposób wskazywać nazwę serwisu i wewnętrzny port w kontenerze, z którym musimy się połączyć. Gdy później dokonamy jakichś zmian, nazwy te nadal będą wskazywały na odpowiedni kontener w naszym stosie. Jest to również wygodne dlatego, że dla czytającego taki plik dość oczywiste staje się, jakie zależności ma dany kontener.

```
environment:
  PORT: 3000
  ROOT_URL: "http://127.0.0.1:3000"
  MONGO_URL: "mongodb://mongo:27017/rocketchat"
  MONGO_OPLOG_URL: "mongodb://mongo:27017/local"
  MAIL_URL: "smtp://smtp.email"
```

Plik *docker-compose.yaml* może też odwoływać się do zmiennych środowiska za pomocą składni `${<NAZWA_ZMIENNEJ>}`, co umożliwia wykorzystanie poufnych danych bez konieczności zapisywania ich w tym pliku. Nowsze wersje Docker Compose obsługują też pliki *.env* (*https://docs.docker.com/compose/env-le*), które mogą być bardzo przydatne do przechowywania danych poufnych i takich zmiennych środowiska, które np. będą różne dla różnych programistów.

Sekcja `depends_on` definiuje kontener, który musi być uruchomiony, zanim dany kontener zostanie uruchomiony. Standardowo `docker-compose` zapewnia jedynie, że kontener jest uruchomiony, a nie że poprawnie działa; możesz jednak w razie potrzeby użyć funkcjonalności `health-check` zarówno w Dockerze, jak i w Docker Compose.

```
depends_on:
  - mongo
```

Funkcjonalność `health-check` Dockera omawiamy bardziej szczegółowo w rozdziale 6., w podrozdziale „Sprawdzanie stanu kontenera”. Możesz również znaleźć więcej informacji w dokumentacji Dockera (*https://dockr.ly/2MYnLZL*) i Docker Compose (*https://dockr.ly/2wt366J*).

Podsekcja `ports` pozwala Ci zdefiniować wszystkie porty, jakie chcesz mieć zmapowane z kontenera do hosta.

```
ports:
  - 3000:3000
```

Usługa `zmachine` korzysta tylko z jednej „nowej” podsekcji, o nazwie `expose`. Ta sekcja pozwala nam poinformować Dockera, że chcemy udostępnić ten port tylko innym kontenerom w sieci Dockera, a nie hostowi. Dlatego nie podajesz tutaj portu hosta, na który port ten ma być zmapowany.

```
expose:
  - "80"
```

W tym momencie mogłeś zauważyć, że choć udostępniliśmy port dla `zmachine`, nie udostępniliśmy portu dla usługi `mongo`. Nie zaszkodziłoby udostępnić port `mongo`, ale nie musieliśmy tego robić, ponieważ jest on już udostępniony w *Dockerfile* wykorzystanym do utworzenia obrazu kontenera `mongo` (*http://bit.ly/2kkFBn*). To czasem jest odrobinę nieprzejrzyste. Może tutaj pomóc wykonanie polecenia `docker history` dla zbudowanego obrazu.

Użyliśmy tutaj przykładu, który jest wystarczająco złożony, by pokazać Ci część mocy Docker Compose, ale w żadnym wypadku nie wyczerpuje on tematu. Istnieje jeszcze wiele innych rzeczy, jakie możesz skonfigurować w pliku *docker-compose.yaml*, w tym ustawienia zabezpieczeń, ograniczenia zasobów i wiele więcej. Możesz znaleźć dużo szczegółowych informacji na temat konfiguracji dla Compose w oficjalnej dokumentacji Docker Compose (*https://docs.docker.com/compose-file*).

Uruchamianie usług

W pliku YAML skonfigurowaliśmy zestaw usług naszej aplikacji. W ten sposób mówimy Compose, co zamierzamy uruchomić i w jaki sposób należy to skonfigurować. A więc uruchamiamy! Aby wydać nasze pierwsze polecenie Docker Compose, musimy upewnić się, że znajdujemy się w tym samym katalogu, w którym znajduje się plik *docker-compose.yaml* odpowiedni dla naszego systemu operacyjnego.

- Użytkownicy systemów typu Unix oraz OS X uruchamiają:

  ```
  $ cd rocketchat-hubot-demo/compose/unix
  ```

- Użytkownicy systemu Windows uruchamiają:

  ```
  PS C:\> cd rocketchat-hubot-demo\compose\windows
  ```

Jedyną różnicą pomiędzy tymi dwoma plikami *docker-compose.yaml* jest to, że wersja dla Windows nie montuje dysku dla bazy danych MongoDB. Spowodowane jest to tym, że MongoDB nie może poprawnie przechowywać danych w systemie plików systemu Windows. Głównym efektem tego będzie fakt, że gdy usuniesz kontener za pomocą komendy `docker-compose down`, wszystkie dane z instancji MongoDB zostaną utracone.

Gdy znajdziesz się w odpowiednim katalogu, możesz potwierdzić, że konfiguracja jest poprawna, uruchamiając:

```
$ docker-compose config
```

Jeśli wszystko jest w porządku, polecenie wyświetli Twój plik konfiguracyjny. Jeśli pojawi się problem, polecenie wyświetli błąd wraz ze szczegółowym opisem problemu, typu:

```
ERROR: The Compose file './docker-compose.yaml' is invalid because:
Unsupported config option for services.hubot: 'networkss'
```

Możesz zbudować wszystkie potrzebne kontenery za pomocą opcji `build`. Wszystkie usługi korzystające z obrazów zostaną pominięte.

```
$ docker-compose build
Building mongo
...
Successfully built a7b0d2b7b1b9
Successfully tagged spkane/mongo:3.2
mongo-init-replica uses an image, skipping
rocketchat uses an image, skipping
zmachine uses an image, skipping
hubot uses an image, skipping
```

Możesz uruchomić swój serwis w tle, uruchamiając poniższe polecenie:

```
$ docker-compose up -d
Creating network "unix_botnet" with driver "bridge"
Creating unix_mongo-init-replica_1 ... done
Creating unix_rocketchat_1 ...
Creating unix_zmachine_1 ... done
Creating unix_zmachine_1 ...
Creating unix_hubot_1 ... done
```

Docker Compose poprzedza nazwy sieci i kontenerów nazwą katalogu zawierającego plik *docker-compose.yaml*. W systemie Windows po wydaniu polecenia zobaczysz więc `windows_botnet`, `windows_mongo-init-replica_1` itd., a nie tekst zamieszczony powyżej.

Użytkownicy Windows: Gdy pierwszy raz uruchomisz usługi, system Windows najprawdopodobniej poprosi Cię o autoryzację `vpnkit`, a Docker for Windows poprosi o udostępnienie Twojego dysku. Musisz w obu przypadkach zezwolić na udostępnienie, zarówno sieci, jak i udziałów dyskowych, aby wszystko uruchomiło się i działało poprawnie.

Gdy wszystko wystartuje, możemy szybko zerknąć na logi wszystkich usług:

```
$ docker-compose logs
...
rocketchat_1 |   | Platform: linux |
mongo_1 | 2018-02-05T01:05:40.200+0000 I REPL [Repli ...
rocketchat_1 |   | Process Port: 3000 |
mongo_1 | 2018-02-05T01:05:40.200+0000 I REPL [Repli ...
rocketchat_1 |   | Site URL: http://127.0.0.1:3000 |
hubot_1 | npm info preuninstall hubot-diagnostics@0.0.2
mongo_1 | 2018-02-05T01:05:40.200+0000 I REPL [Repli ...
rocketchat_1 |   | ReplicaSet OpLog: Enabled |
...
```

Nie zobaczysz tego tutaj, ale jeśli uruchamiasz przykład podczas czytania tego tekstu, zauważ, że logi są oznaczone różnymi kolorami dla różnych serwisów i wyświetlane w takiej kolejności, w jakiej Docker je otrzymuje. To bardzo ułatwia śledzenie, co się dzieje, nawet w sytuacji, gdy wiele serwisów generuje logi w tym samym czasie.

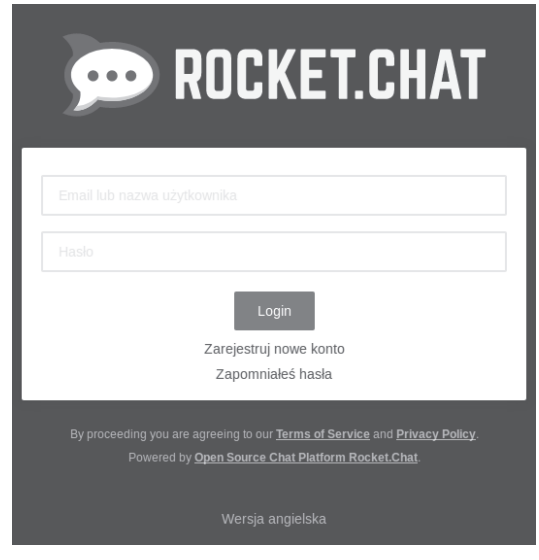
W tym momencie udało nam się uruchomić dość złożoną aplikację, która tworzy stos kontenerów. Zerkniemy teraz na tę prostą aplikację, abyś mógł zobaczyć, co zbudowaliśmy, i dokładniej zrozumieć działanie narzędzi Compose. Choć kolejny podrozdział nie zawiera nic ściśle związanego z samym Dockerem, ma on na celu pokazanie Ci, jak łatwo można wykorzystać Docker Compose do uruchomienia złożonego i w pełni funkcjonalnego serwisu sieciowego.

Poznajemy RocketChat

W tym podrozdziale na chwilę oddalimy się od Dockera i przyjrzymy się RocketChat. Poświęcimy kilka stron na to, byś dowiedział się wystarczająco wiele, by zacząć doceniać, o ile prostsze jest skonfigurowanie złożonego środowiska za pomocą Docker Compose. Jeśli chcesz, możesz przeskoczyć do kolejnego podrozdziału, „Ćwiczenia z Docker Compose”.

Krótko zerkniemy, co dzieje się w środku tego, co przygotowaliśmy. Aby jednak było to efektywne, musimy teraz poświęcić chwilę na poznanie stosu aplikacji, jaki zbudowaliśmy. Aplikacja uruchomiona za pomocą Docker Compose, RocketChat, jest oprogramowaniem typu klient/serwer o otwartych źródłach. Aby zobaczyć, w jaki sposób ona działa, uruchom przeglądarkę internetową i przejdź pod adres *http://127.0.0.1:3000*.

Gdy się tam znajdziesz, zobaczysz ekran logowania do RocketChat (*https://rocket.chat/*) (rysunek 8.1).



Rysunek 8.1. Ekran logowania do RocketChat

Kliknij odnośnik *Zarejestruj nowe konto* i wypełnij formularz danymi:

- Nazwa: **`admin`**
- E-mail: **`admin@przyklad.com`**
- Hasło: **`p2sswOrd`**
- Hasło: **`p2sswOrd`**

Następnie wybierz duży niebieski przycisk opisany tekstem *Zarejestruj nowe konto* (rysunek 8.2).

Rysunek 8.2. Ekran rejestracji RocketChat

Teraz musisz zaczekać. Tworzenie konta zajmie przynajmniej minutę, w czasie której serwer ustawia wszystko po raz pierwszy. Spróbuj powstrzymać się przed kliknięciem niebieskiego przycisku, na którym znajduje się teraz napis *Proszę czekaj…*. To nie powinno niczego zepsuć, ale też niczego nie przyspieszy.

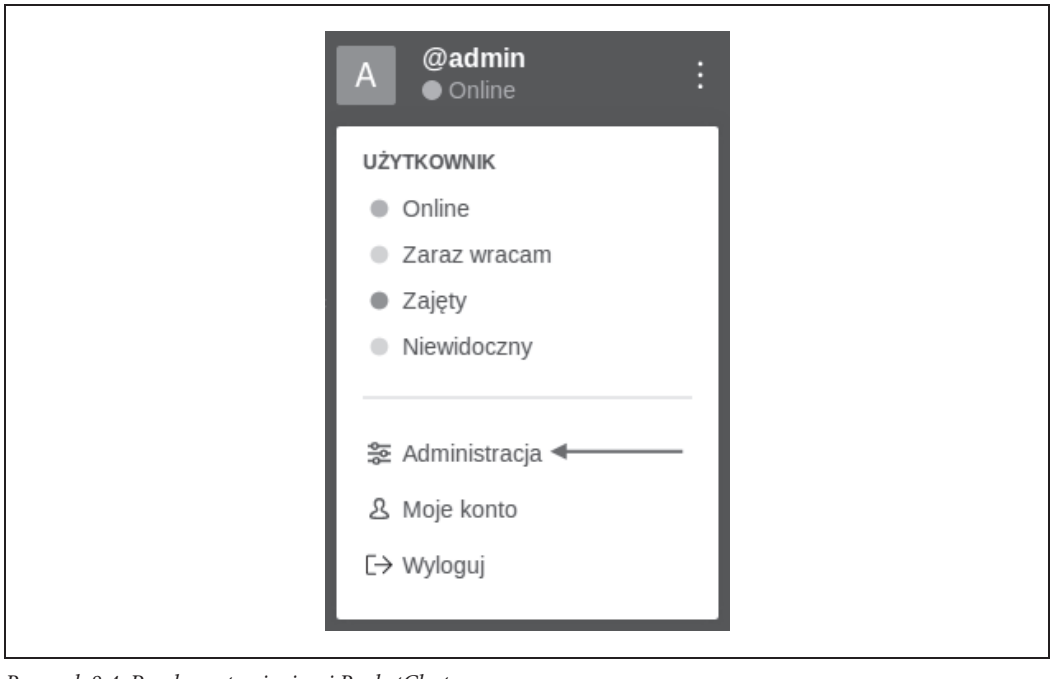
Gdy wszystko będzie już przygotowane, zobaczysz okno dialogowe zatytułowane *Utwórz nazwę użytkownika*. Możesz pozostawić taką nazwę użytkownika, jaka będzie tam wpisana, jeśli nie zechcesz jej zmienić — po prostu kliknij przycisk opisany *Użyj tej nazwy użytkownika* (rysunek 8.3).

Gratulacje — jesteś teraz zalogowany do w pełni funkcjonalnego klienta czatu, ale to jeszcze nie wszystko. Docker Compose uruchomił również instancję asystenta czatu o nazwie hubot (*https://hubot.github.com/*) oraz tajemniczą usługę `zmachine`, a więc przyjrzyjmy się też im.

Ponieważ mamy całkiem nowy serwer RocketChat, nie istnieje w nim jeszcze użytkownik, którego mógłby użyć nasz bot. Zmieńmy to.

Rysunek 8.3. Ekran rejestracji nazwy użytkownika RocketChat

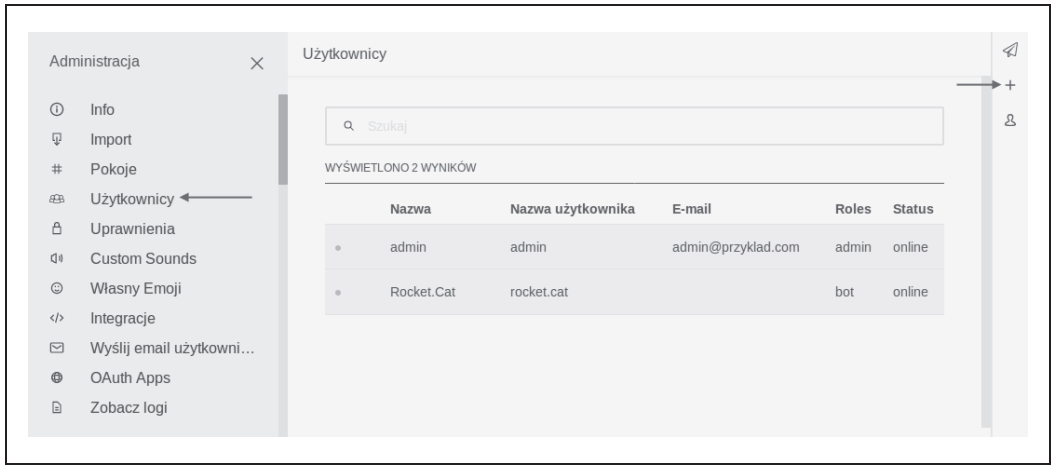
Zacznij, klikając na górze paska znajdującego się po lewej stronie w miejscu, gdzie widzisz nazwę swojego użytkownika i status. Spowoduje to wyświetlenie menu, w którym będzie możliwe wybranie opcji *Administracja* (rysunek 8.4).



Rysunek 8.4. Pasek z ustawieniami RocketChat

Podczas testowania zauważyliśmy, że niektóre przeglądarki internetowe lub systemy operacyjne nie zawsze wyświetlają mniejsze ikony w interfejsie RocketChat. Były one klikalne, ale było trudno stwierdzić, że tam się cokolwiek znajduje. Jeśli widzisz coś takiego, przyciski nadal działają, a powolne przesuwanie wskaźnika myszy nad nimi powinno spowodować wyświetlenie podpowiedzi.

Lewy pasek po kliknięciu zostanie zastąpiony panelem administracyjnym. Na tym panelu kliknij *Użytkownicy* (rysunek 8.5).

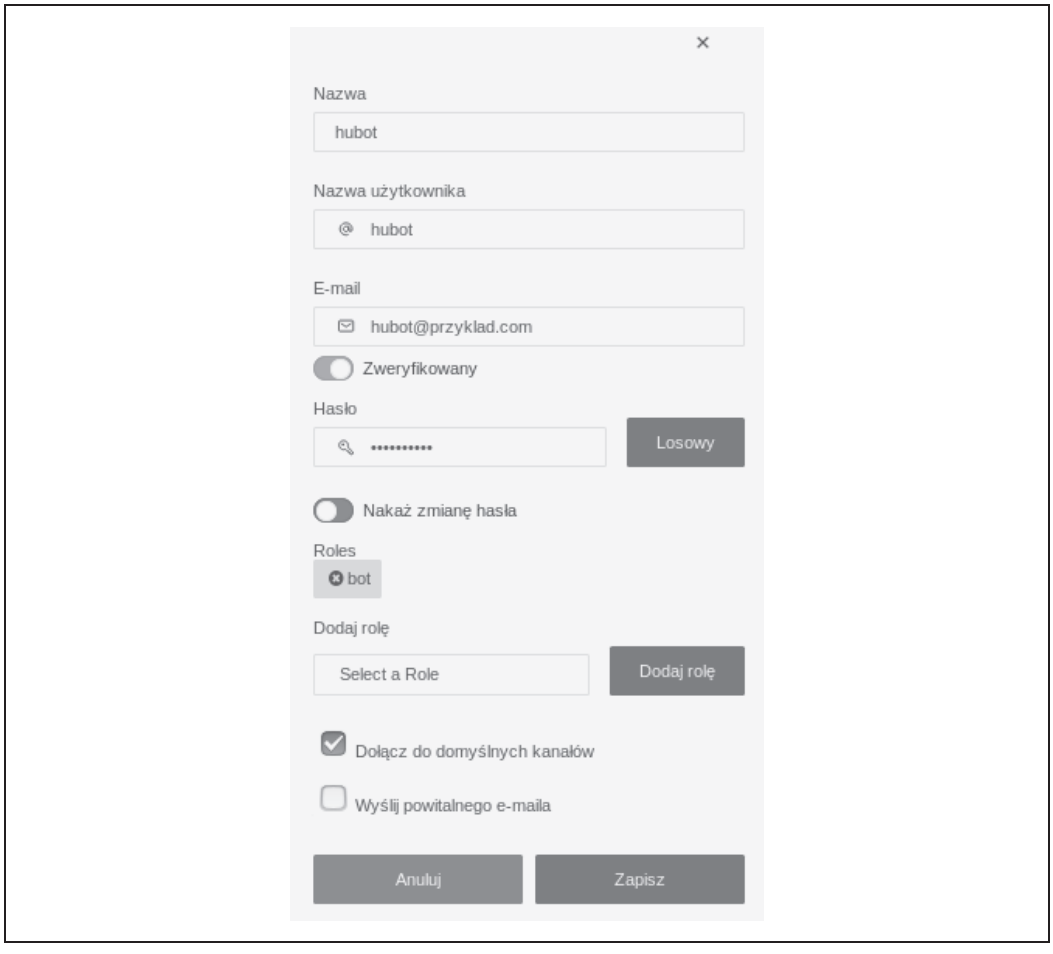


Rysunek 8.5. Ekran umożliwiający dodawanie użytkownika w RocketChat

Po prawej stronie ekranu kliknij symbol +, aby dodać użytkownika, a następnie wypełnij formularz danymi:

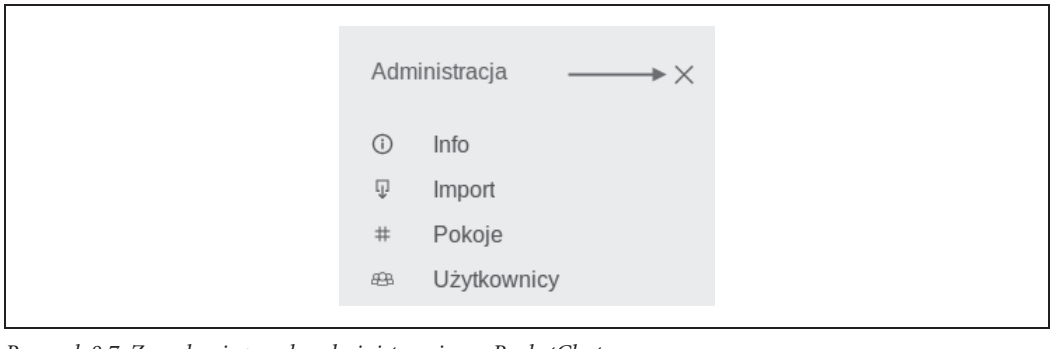
- Nazwa: **`hubot`**
- Nazwa użytkownika: **`hubot`**
- E-mail: **`hubot@przyklad.com`**
- Przełącz: **`Zweryfikowany (Zielone)`**
- Hasło: **`HughTheBot`**
- Przełącz: **`Nakaż zmianę hasła (Czerwone)`**
- Wybierz rolę: **`bot`**
- Przycisk: **`Dodaj rolę`**
- Odznacz: **`Wyślij powitalnego e-maila`**

Kliknij przycisk *Zapisz* na dole, aby utworzyć użytkownika (rysunek 8.6).



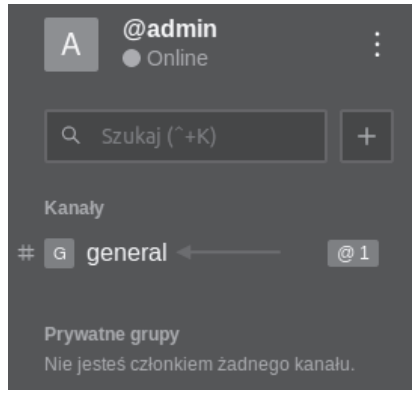
Rysunek 8.6. Ekran pozwalający na utworzenie w aplikacji RocketChat konta użytkownika dla bota

Na górze po lewej stronie panelu administracyjnego kliknij znak *X*, aby zamknąć panel (rysunek 8.7).



Rysunek 8.7. Zamykanie panelu administracyjnego RocketChat

W panelu po lewej stronie kliknij *general* (rysunek 8.8).



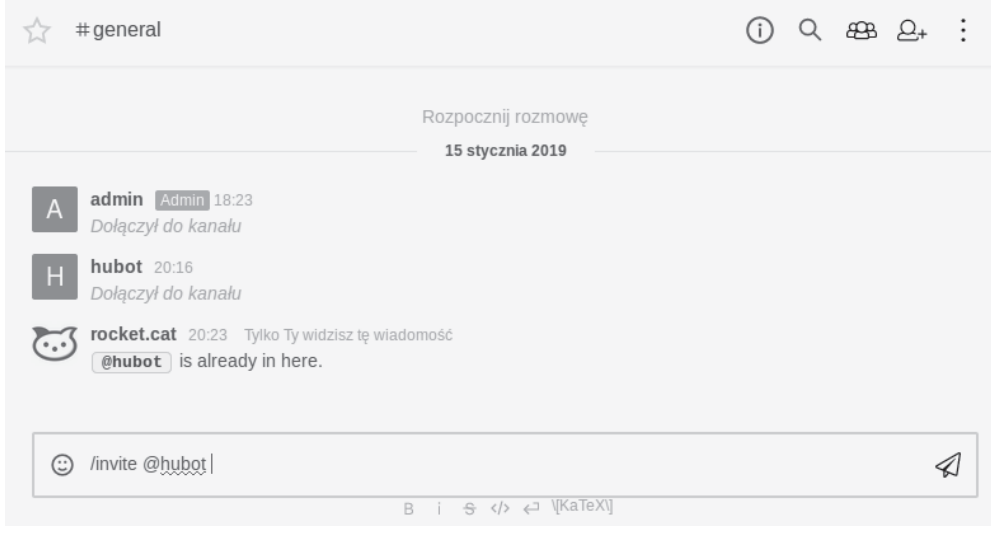
Rysunek 8.8. Kanał general w RocketChat

W końcu spróbujmy podłączyć hubot do serwera czatu. Najprościej można to zrobić, restartując kontener hubot za pomocą Docker Compose w terminalu.

```
$ docker-compose restart hubot
Restarting unix_hubot_1 ... done
```

Jeśli wszystko pójdzie zgodnie z planem, powinieneś w tym momencie móc wrócić do swojej przeglądarki internetowej i przesyłać polecenia do Twojego bota w oknie czatu.

W polu wiadomości na dole ekranu z chatem kanału *general* zacznij od wpisania `/invite @hubot`, aby upewnić się, że nasz bot śledzi wiadomości tego kanału (rysunek 8.9).



Rysunek. 8.9. Zapraszanie hubot w RocketChat

Możesz otrzymać od wewnętrznego administratora `rocket.cat` wiadomość o treści `@hubot is already here`. To bardzo dobrze.

Zmienne środowiska użyte do konfigurowania bota hubot przypisały mu alias w postaci kropki. Możesz więc teraz spróbować wpisać `. help`, aby sprawdzić, czy bot odpowiada. Jeśli wszystko działa, powinieneś otrzymać listę poleceń, które bot rozpoznaje i na które będzie odpowiadał.

```
> . help
. adapter - Reply with the adapter
. echo <text> - Reply back with <text>
. help - Displays all of the help commands that this bot knows about.
. help <query> - Displays all help commands that match <query>.
...
. meme One does not simply <text> - Lord of the Rings Boromir
...
. ping - Reply with pong
. time - Reply with current time
. z <action> - Performs the action
...
. z stop - Ends the game for this channel
```

Spróbuj więc wpisać tekst:

```
. meme One does not simply launch a complex web service with docker-compose!
```

Na koniec spróbuj utworzyć nowy kanał, wpisując `/create zmachine` w oknie czatu. Dzięki temu powinieneś móc kliknąć na nazwę nowego kanału na lewym pasku i zaprosić hubot na ten kanał za pomocą polecenia `/invite @hubot`.

Gdy to zrobisz, hubot może powiedzieć:

```
There's no game for zmachine!
```

Nie ma się czym przejmować.

Wpisując podane niżej komendy w oknie chata, możesz zagrać w przystosowaną do czatu wersję sławnej gry *Colossal Cave Adventure* (*https://pl.wikipedia.org/wiki/Colossal_Cave_Adventure*):

```
. z start adventure

more
look
go east
examine keys
get keys

. z save firstgame
. z stop
. z start adventure
. z restore firstgame

inventory
```

Gry tego typu mogą być uzależniające i można stracić wiele czasu. Ostrzegamy... Pamiętaj o tym, ale jeśli nie znasz tego tematu, a chcesz dowiedzieć się więcej, poniżej znajdziesz kilka odnośników do przejrzenia.

- Definiton of Interactive Fiction (*http://bit.ly/2PRkLNJ*).
- Emulator (*http://frotz.sourceforge.net/*).
- Development (*http://inform7.com/download*).
- Games (*http://bit.ly/2LCI5LL*).
- Competition (*https://ifcomp.org/*).

Zobaczyłeś już, jak proste może być skonfigurowanie, uruchomienie i zarządzanie za pomocą Docker Compose złożonymi serwisami internetowymi, które wymagają do działania wielu komponentów. W kolejnym podrozdziale przyjrzymy się kilku kolejnym możliwościom Docker Compose.

Ćwiczenia z Docker Compose

Mając już uruchomioną aplikację i wiedząc, co ona robi, możemy zajrzeć głębiej, by odrobinę lepiej poznać, w jaki sposób te usługi działają. Niektóre z poleceń Dockera mają swoje odpowiedniki w Compose, ale wykonują operacje na wskazanym stosie, a nie na pojedynczych kontenerach ani wszystkich kontenerach hosta. Możesz uruchomić `docker-compose top`, by zobaczyć podsumowanie informacji o swoich kontenerach i procesach w nich uruchomionych.

```
$ docker-compose top
unix_hubot_1
PID  USER TIME                                COMMAND
------------------------------------------------------------------------...
1353 1001 0:00 /bin/sh -c node -e
               "console.log(JSON.stringify('$EXTERNAL_SCRIPTS'.split...
               external-scripts.json && npm install $(node -e
               "console.log('$EXTERNAL_SCRIPTS'.split(',').join(' ')...
               bin/hubot
               -n $BOT_NAME -a rocketchat
1521 1001 0:02 node node_modules/.bin/coffee node_modules/.bin/hubot...
               -n bot -a rocketchat

unix_mongo_1
 PID   USER   TIME                  COMMAND
------------------------------------------------------------------
23104  999    0:46    mongod --smallfiles --oplogSize 128 --replSet rs0

unix_rocketchat_1
 PID  USER  TIME   COMMAND
-----------------------------------
23399 99999 0:34 node main.js

unix_zmachine_1
PID   USER TIME                          COMMAND
------------------------------------------------------------------------...
2808  0    0:00 /root/src/../frotz/dfrotz -S 0 /root/src/../zcode/ad...
23710 0    0:00 node /root/src/server.js
```

Podobnie jak przy uruchamianiu kontenera Dockera, możesz uruchomić polecenia wewnątrz kontenerów przy użyciu narzędzia Docker Compose, wydając polecenie `docker-compose exec`:

```
$ docker-compose exec mongo bash
root@6f12015f1dfe:/# mongo
MongoDB shell version: 3.2.18
connecting to: test
Welcome to the MongoDB shell.
For interactive help, type "help".
For more comprehensive documentation, see
http://docs.mongodb.org/
Questions? Try the support group
http://groups.google.com/group/mongodb-user
rs0:PRIMARY> exit
bye
root@6f12015f1dfe:/# exit
```

`docker-compose logs` oraz `docker-compose exec` to prawdopodobnie najbardziej przydatne polecenia przy usuwaniu problemów. Jeśli `docker-compose` nie może zbudować obrazu lub uruchomić kontenera, może być konieczne skorzystanie ze standardowych poleceń Dockera, by usunąć błędy z obrazu i kontenera, co omówiliśmy w podrozdziale „Usuwanie problemów z obrazami” w rozdziale 4. oraz „Wnętrze działającego kontenera” w rozdziale 6.

Możesz też użyć Docker Compose, aby uruchomić (ang. *start*), zatrzymać (ang. *stop*), a w większości środowisk również wstrzymać (ang. *pause*) i uruchomić ponownie (ang. *unpause*) zarówno pojedynczy kontener, jak i wszystkie swoje kontenery, w zależności od potrzeb.

```
$ docker-compose stop zmachine
Stopping unix_zmachine_1 ... done
$ docker-compose start zmachine
Starting zmachine ... done
$ docker-compose pause
Pausing unix_mongo_1      ... done
Pausing unix_rocketchat_1 ... done
Pausing unix_zmachine_1   ... done
Pausing unix_hubot_1      ... done
$ docker-compose unpause
Unpausing unix_hubot_1      ... done
Unpausing unix_zmachine_1   ... done
Unpausing unix_rocketchat_1 ... done
Unpausing unix_mongo_1      ... done
```

Gdy na koniec zechcesz wszystko zatrzymać i usunąć wszystkie utworzone przez `docker-compose` kontenery, możesz uruchomić następujące polecenie:

```
$ docker-compose down
Stopping unix_hubot_1      ... done
Stopping unix_zmachine_1   ... done
Stopping unix_rocketchat_1 ... done
Stopping unix_mongo_1      ... done
Removing unix_hubot_1              ... done
Removing unix_zmachine_1           ... done
Removing unix_rocketchat_1         ... done
Removing unix_mongo-init-replica_1 ... done
Removing unix_mongo_1              ... done
Removing network unix_botnet
```

Użytkownicy Windows: Gdy skasujecie kontener MongoDB za pomocą polecenia `docker-compose down`, wszystkie dane z instancji MongoDB zostaną utracone.

Podsumowanie

Powinieneś w tym momencie bardzo dobrze rozumieć, co możesz osiągnąć dzięki Docker Compose i w jaki sposób można wykorzystać to narzędzie, by zmniejszyć uciążliwość i zwiększyć powtarzalność tworzenia swoich środowisk deweloperskich.

W kolejnym rozdziale poznamy narzędzia przydatne do wspomagania skalowania Dockera w centrum przetwarzania danych oraz w chmurze.

Skorowidz

Ś

T

U

V

W

Z

O'REILLY®

Koniec problemów z zależnościami w aplikacjach!

Docker jest nową technologią, która radykalnie zmieniła podejście do wdrażania oprogramowania. Obrazy i kontenery Dockera upraszczają zarządzanie zależnościami, dzięki czemu testowanie, wdrażanie i skalowanie aplikacji staje się o wiele prostsze. W ciągu ostatnich lat Docker znacznie się rozwinął. Jest teraz bardzo stabilny i daje programistom wiele narzędzi do wyboru. W oczywisty sposób zwiększa to jego popularność wśród twórców dużych systemów. Niemniej zrozumieć działanie Dockera i nauczyć się wykorzystywać go w poprawny sposób — to nie jest trywialne i wymaga wysiłku.

Oto kolejne — zaktualizowane i uzupełnione — wydanie praktycznego przewodnika, dzięki któremu szybko nauczysz się korzystać z Dockera. Wyjaśniono tu podstawy jego działania, pokazano praktyczne techniki wdrażania i testowania kontenerów Dockera, przedstawiono także podstawowe wewnętrzne procesy kontenerów. Z książki dowiesz się, jak przygotować pakiet aplikacji ze wszystkimi ich zależnościami, a następnie przetestować go, wdrożyć, skalować oraz utrzymywać w środowiskach produkcyjnych. Znajdziesz tu również nowy rozdział na temat Docker Compose, głębsze omówienie trybu Docker Swarm, wprowadzenie do Kubernetes, a także przykłady optymalizacji obrazów Dockera i wiele innych przydatnych informacji.

Najważniejsze zagadnienia:

- solidne wprowadzenie do Dockera oraz przygotowanie środowiska pracy
- debugowanie obrazów i kontenerów Dockera
- sprawne wdrażanie aplikacji w środowiskach produkcyjnych
- wdrażanie kontenerów w publicznych i prywatnych chmurach

Sean P. Kane — ma wieloletnie doświadczenie w utrzymywaniu aplikacji produkcyjnych dla wielu branż przemysłowych. Swój wolny czas poświęca na pisanie, uczenie i podróże.

Karl Matthias — jest znakomitym programistą, architektem systemów rozproszonych oraz inżynierem sieci. Uwielbia stare kamery i jazdę na rowerze.

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

ISBN 978-83-283-5604-7

Cena: 59,00 zł